Scilab Textbook Companion for Electrical Power Systems: Concepts, Theory and Practice
by S. Ray[1]

Created by
Jai Mathur
MCA
Computer Engineering
IIT Bombay
College Teacher
None
Cross-Checked by
Bhavani Jalkrish

July 14, 2017

[1]Funded by a grant from the National Mission on Education through ICT, http://spoken-tutorial.org/NMEICT-Intro. This Textbook Companion and Scilab codes written in it can be downloaded from the "Textbook Companion Project" section at the website http://scilab.in

Book Description

Title: Electrical Power Systems: Concepts, Theory and Practice

Author: S. Ray

Publisher: Prentice Hall, New Delhi

Edition: 1

Year: 2007

ISBN: 9788120329898

Scilab numbering policy used in this document and the relation to the above book.

Exa Example (Solved example)

Eqn Equation (Particular equation of the above book)

AP Appendix to Example(Scilab Code that is an Appednix to a particular Example of the above book)

For example, Exa 3.51 means solved example 3.51 of this book. Sec 2.3 means a scilab code whose theory is explained in Section 2.3 of the book.

Contents

List of Scilab Codes 4

2 FUNDAMENTAL CONCEPTS OF AC CIRCUITS 5

3 GENERAL CONSIDERATIONS OF TRANSMISSION AND DISTRIBUTION 8

4 ELECTRICAL CHARACTERISTICS MODELLING AND PERFORMANCE OF AERIAL TRANSMISSION LINES 11

5 OVERHEAD LINE CONSTRUCTION 36

6 UNDERGROUND CABLES 50

7 SUBSTATION AND DISTRIBUTION SYSTEM 54

8 ELEMENTS OF ELECTRIC POWER GENERATION 68

9 LOAD FLOW STUDIES 76

10 POWER SYSTEM ECONOMICS 81

12 OVER VOLTAGE TRANSIENTS IN POWER SYSTEMS AND PROTECTION 86

13 SHORT CIRCUIT PHENOMENA 91

14 ELEMENTS OF CIRCUIT BREAKERS AND RELAYS 114

15 POWER SYSTEM STABILITY 119

List of Scilab Codes

Exa 2.1 Example . . . 5
Exa 2.2 Example . . . 6
Exa 3.1 Example . . . 8
Exa 4.1 Example . . . 11
Exa 4.2 Example . . . 12
Exa 4.3 Example . . . 12
Exa 4.4 Example . . . 13
Exa 4.5 Example . . . 14
Exa 4.6 Example . . . 15
Exa 4.7 Example . . . 17
Exa 4.8 Example . . . 17
Exa 4.9 Example . . . 18
Exa 4.10 Example . . . 19
Exa 4.11 Example . . . 20
Exa 4.12 Example . . . 20
Exa 4.13 Example . . . 23
Exa 4.14 Example . . . 24
Exa 4.16 Example . . . 28
Exa 4.17 Example . . . 30
Exa 4.18 Example . . . 31
Exa 4.19 Example . . . 33
Exa 5.1 Example . . . 36
Exa 5.2 Example . . . 38
Exa 5.3 Example . . . 39
Exa 5.5 Example . . . 40
Exa 5.6 Example . . . 41
Exa 5.7 Example . . . 43
Exa 5.8 Example . . . 45

Exa 5.9 Example 46
Exa 5.10 Example 47
Exa 5.11 Example 48
Exa 6.1 Example 50
Exa 6.2 Example 51
Exa 6.3 Example 51
Exa 6.4 Example 53
Exa 7.1 Example 54
Exa 7.2 Example 57
Exa 7.3 Example 59
Exa 7.4 Example 61
Exa 7.5 Example 63
Exa 7.6 Example 64
Exa 7.8 Example 65
Exa 8.1 Example 68
Exa 8.2 Example 68
Exa 8.3 Example 69
Exa 8.4 Example 70
Exa 8.5 Example 70
Exa 8.6 Example 72
Exa 8.7 Example 73
Exa 8.8 Example 74
Exa 9.1 Example 76
Exa 9.2 Example 79
Exa 10.1 Example 81
Exa 10.2 Example 82
Exa 10.3 Example 83
Exa 12.1 Example 86
Exa 12.2 Example 87
Exa 12.3 Example 88
Exa 12.5 Example 89
Exa 12.7 Example 89
Exa 13.1 Example 91
Exa 13.2 Example 94
Exa 13.3 Example 95
Exa 13.4 Example 96
Exa 13.5 Example 98
Exa 13.6 Example 99

Exa 13.7 Example 101
Exa 13.8 Example 101
Exa 13.9 Example 106
Exa 13.10 Example 107
Exa 13.11 Example 110
Exa 14.1 Example 114
Exa 14.2 Example 115
Exa 14.3 Example 117
Exa 14.4 Example 118
Exa 15.1 Example 119
Exa 15.2 Example 120
Exa 15.3 Example 121
Exa 15.4 Example 123
Exa 15.5 Example 125

Chapter 2

FUNDAMENTAL CONCEPTS OF AC CIRCUITS

Scilab code Exa 2.1 Example

```

// Variable Declaration
MVA_base = 10.0  //Three-phase base MVA
kV_base = 13.8   //Line-line base kV
P = 7.0          //Power delivered(MW)
PF = 0.8         //Power factor lagging
Z = 5.7          //Impedance(ohm)

// Calculation Section
I_base = (MVA_base) * (10**3)/((3**(0.5)) * kV_base)      //Base current(A)
I_actual = P * (10**3)/((3**(0.5)) * kV_base*PF)          //Actual current delivered by machine(A)
I_pu = I_actual/I_base                                    //p.u current(p.u)
Z_pu = Z * (MVA_base/( (kV_base)**2 ))                    //p.u impedance(p.u)
P_act_pu = P/MVA_base
```

```
//p.u active power(p.u)
x = acos(PF)
y = sin(x)
P_react = (P * y)/PF //Actual reactive power(MVAR)
P_react_pu = P_react/MVA_base //Actual p.u reactive power(p.u)

// Result Section
printf('p.u current = %.3f p.u',I_pu)
printf('p.u impedance = %.1f p.u',Z_pu)
printf('p.u active power = %.1f p.u',P_act_pu)
printf('p.u reactive power = %.3f p.u',P_react_pu)
```

Scilab code Exa 2.2 Example

```

// Variable Declaration
MVA_base = 5.0    //Base MVA on both sides
hv_base = 11.0    //Line to line base voltages in kV on h.v side
lv_base = 0.4     //Line to line base voltages in kV on l.v side
Z = 5.0/100       //Impedance of 5%

// Calculation Section
Z_base_hv = (hv_base)**2/MVA_base    //Base impedance on h.v side(ohm)
Z_base_lv = (lv_base)**2/MVA_base    //Base impedance on l.v side(ohm)
Z_act_hv = Z * Z_base_hv             //Actual impedance viewed from h.v side(ohm)
```

```
Z_act_lv = Z * Z_base_lv              //Actual
   impedance viewed from l.v side(ohm)

// Result Section
printf('Base impedance on h.v side = %.1f ohm',
   Z_base_hv)
printf('Base impedance on l.v side = %.3f ohm',
   Z_base_lv)
printf('Actual impedance viewed from h.v side = %.2f
    ohm',Z_act_hv)
printf('Actual impedance viewed from l.v side = %.4f
    ohm',Z_act_lv)
```

Chapter 3

GENERAL CONSIDERATIONS OF TRANSMISSION AND DISTRIBUTION

Scilab code Exa 3.1 Example

```

// Variable Declaration
P = 5.0                  //Power(MW)
pf = 0.8                 //lagging power factor
d = 15.0                 //Distance of line(km)
J = 4.0                  //Current density(amp per mm^2)
r = 1.78*10**(-8)        //Resistivity(ohm-m)
kV_1 = 11.0              //Permissible voltage level(kV)
kV_2 = 22.0              //Permissible voltage level(kV)

// Calculation Section
I_1 = (P*10**3)/((3)**(0.5) * (kV_1) * pf)    //Load
    current(A)
area_1 = I_1/J                                  //Cross
    -sectional area of the phase conductor(mm^2)
```

```
volume_1 = 3 * (area_1/10**6) * (d*10**3)     //Volume of conductors material(m^3)
R_1 = r * (d*10**3)/(area_1 * (10**-6))       //Resistance per phase(ohm)
PL_1 = 3 * (I_1**2) * (R_1*10**(-3))          //Power loss(kW)

I_2 = (P*10**3)/((3)**(0.5) * (kV_2) * pf)    //Load current(A)
area_2 = I_2/J                                //Cross-sectional area of the phase conductor(mm^2)
volume_2 = 3 * (area_2/10**6) * (d*10**3)     //Volume of conductors material(m^3)
R_2 = r * (d*10**3)/(area_2 * (10**-6))       //Resistance per phase(ohm)
PL_2 = 3 * (I_2**2) * (R_2*10**(-3))          //Power loss(kW)
area_ch = (area_1-area_2)/area_1*100          //Change in area of 22kV level from 11 kV level(%)
vol_ch = (volume_1-volume_2)/volume_1*100     //Change in volume of 22kV level from 11 kV level(%)
loss_ch = (PL_1-PL_2)/PL_1*100                //Change in losses of 22kV level from 11 kV level(%)

// Result Section
printf('For 11 kV level :')
printf('Cross-sectional area of the phase conductor = %d mm^2' ,area_1)
printf('Volume of conductors material = %.2f m^3' ,volume_1)
printf('Power loss = %.2f kW' ,PL_1)
printf('\nFor 22 kV level :')
printf('Cross-sectional area of the phase conductor = %d mm^2' ,area_2)
printf('Volume of conductors material = %.2f m^3' ,volume_2)
```

```
printf('Power loss = %.2f kW',PL_2)
printf('\nConductor size has decreased by %.f
    percent in 22 kV level',area_ch)
printf('Conductor volume has decreased by %.f
    percent in 22 kV level',vol_ch)
printf('Conductor losses has decreased by %.f
    percent in 22 kV level',loss_ch)
```

Chapter 4

ELECTRICAL CHARACTERISTICS MODELLING AND PERFORMANCE OF AERIAL TRANSMISSION LINES

Scilab code Exa 4.1 Example

```

// Variable Declaration
l = 10.0         //Length of 1-phase line(km)
d = 100.0        //Spacing b/w conductors(cm)
r = 0.3          //Radius(cm)
u_r_1 = 1.0      //Relative permeability of copper
u_r_2 = 100.0    //Relative permeability of steel

// Calculation Section
r_1 = 0.7788*r                                  //
   Radius of hypothetical conductor(cm)
L_1 = 4 * 10**(-7) * log(d/r_1)             //Loop
   inductance(H/m)
```

```
L_T1 = L_1 * l * 10**6                          //Total loop inductance(mH)

L_2 = 4 * 10**(-7) * (log(d/r) + (u_r_2/4))//Loop inductance(H/m)
L_T2 = L_2 * l * 10**6                          //Total loop inductance(mH)

// Result Section
printf('(i) Total loop inductance of copper conductor = %.2f mH' ,L_T1)
printf('(ii)Total loop inductance of steel conductor = %.2f mH' ,L_T2)
```

Scilab code Exa 4.2 Example

```
// Variable Declaration
r = 0.4     //Radius of conductor(cm)
h = 1000    //Height of line(cm)

// Calculation Section
d = 2*h                                          //Spacing between conductors(cm)
L = 2 * 10**(-4) * log(2*h/(0.7788*r)) * 1000    //Inductance of conductor(mH/km)

// Result Section
printf('Inductance of the conductor = %.3f mH/km' ,L)
```

Scilab code Exa 4.3 Example

```
// Variable Declaration
d_ab = 4      //Distance b/w conductor a & b(m)
d_bc = 9      //Distance b/w conductor b & c(m)
d_ca = 6      //Distance b/w conductor c & a(m)
r = 1.0       //Radius of each conductor(cm)

// Calculation Section
D_m = (d_ab * d_bc * d_ca)**(1.0/3)              //Geometric mean separation(m)
r_1 = 0.7788 * (r/100)                           //Radius of hypothetical conductor(m)
L = 2 * 10**(-7) * log(D_m/r_1) * 10**6     //Line inductance(mH/phase/km)

// Result Section
printf('Line inductance , L = %.2f mH/phase/km' ,L)
```

Scilab code Exa 4.4 Example

```
// Variable Declaration
r = 1.0         //Radius of each conductor(cm)
d_11 = 30       //Distance b/w conductor 1 & 1'(cm)
d_22 = 30       //Distance b/w conductor 2 & 2'(cm)
d_12 = 130      //Distance b/w conductor 1 & 2(cm)
d_122 = 160     //Distance b/w conductor 1 & 2'(cm)
d_112 = 100     //Distance b/w conductor 1' & 2(cm)
d_1122 = 130    //Distance b/w conductor 1' & 2'(cm)

// Calculation Section
r_1 = 0.7788 * r                                 //Radius of hypothetical conductor(cm)
D_s = (d_11 * r_1 * d_22 * r_1)**(1.0/4)         //Geometric mean radius(cm)
```

```
D_m = (d_12 * d_122 * d_112 * d_1122)**(1.0/4) // Geometric mean separation(cm)
L = 4 * 10**(-7) * log(D_m/D_s) * 10**6    //Loop inductance(mH/km)

R = 2**0.5                                      // Radius of single conductor(cm)
d = 130.0                                       // Conductor position(cm)
L_1 = 4*10**(-7)*log(d/(0.7788*R))*10**6   //Loop inductance(mH/km)
L_diff = (L_1 - L)/L*100                        // Change in inductance(%)
r_diff = D_s - R                                // Effective radius difference


// Result Section
printf('Loop inductance , L = %.3f mH/km' ,L)
printf('Loop inductance having two conductors only , L = %.3f mH/km' ,L_1)
printf('There is an Increase of %.f percent in inductance value ' ,L_diff)
printf('Effective radius of bundled conductors is about %.1f times that of unbundled system reducing field stress almost by that ratio' , r_diff)
```

Scilab code Exa 4.5 Example

```

// Variable Declaration
r = 1.5           //Radius of each conductor(cm)
D_a1a2 = 0.3      //Distance b/w conductor a1 & a2(m)
D_a2a1 = 0.3      //Distance b/w conductor a2 & a1(m)
```

```
D_a1b1 = 15.3   //Distance b/w conductor a1 & b1(m)
D_a1b2 = 15.6   //Distance b/w conductor a1 & b2(m)
D_a2b1 = 15.0   //Distance b/w conductor a2 & b1(m)
D_a2b2 = 15.3   //Distance b/w conductor a2 & b2(m)
D_b1c1 = 15.3   //Distance b/w conductor b1 & c1(m)
D_b1c2 = 15.6   //Distance b/w conductor b1 & c2(m)
D_b2c1 = 15.0   //Distance b/w conductor b2 & c1(m)
D_b2c2 = 15.3   //Distance b/w conductor b2 & c2(m)
D_a1c1 = 30.6   //Distance b/w conductor a1 & c1(m)
D_a1c2 = 30.9   //Distance b/w conductor a1 & c2(m)
D_a2c1 = 30.3   //Distance b/w conductor a2 & c1(m)
D_a2c2 = 30.6   //Distance b/w conductor a2 & c2(m)

// Calculation Section
r_1 = 0.7788 * (r/100)
    //Radius of hypothetical conductor(m)
D_s = (D_a1a2 * r_1 * D_a2a1 * r_1)**(1.0/4)
    //Geometric mean radius(m)
D_ab = (D_a1b1 * D_a1b2 * D_a2b1 * D_a2b2)**(1.0/4)
    //Mutual GMD b/w conductor a & b(m)
D_bc = (D_b1c1 * D_b1c2 * D_b2c1 * D_b2c2)**(1.0/4)
    //Mutual GMD b/w conductor b & c(m)
D_ca = (D_a1c1 * D_a1c2 * D_a2c1 * D_a2c2)**(1.0/4)
    //Mutual GMD b/w conductor c & a(m)
D_m = (D_ab * D_bc * D_ca)**(1.0/3)
    //Geometric mean separation(m)
L = 2 * 10**(-4) * log(D_m/D_s) * 1000          //
    Inductance(mH/km)

// Result Section
printf('Inductance/phase/km = %.3f mH/km' ,L)
```

Scilab code Exa 4.6 Example

```

```

```
// part - I
// Dsa = GMR of phase a in section - I
// (r'Da1a2)(Da1a2r')^(1/4) = sqrt(r'Da1a2)
// Da1a2 = sqrt(D^2 +  4d^2)
printf(" Dsa = sqrt(r * sqrt(D^2 + 4*d^2))")

// Dsb = GMR of phase b in section - II
// Dsb = sqrt(r * Db1b2)
// Db1b2 = D

printf(" Dsb = sqrt(rD) ")

// Dsc = GMR of phase c in section - I
//  = sqrt(r'Dc1c2)
// Dc1c2 = sqrt(D^2 + rd^2)
printf(" Dsc = sqrt(r * sqrt(D^2 + 4*d^2))")

// part - II
// Dab = Mutual GMD between phase a and b in section
    I of the trasportation cycle.

printf(" Dab = sqrt(d * sqrt(d^2 + D^2))")
printf(" Dbc = sqrt(d * sqrt(d^2 + D^2))")
printf(" Dca = sqrt(2d * D)")

// part - III
// GMD for fictitious equilateral spacing

printf ( " Ds = (r)^(1/2) * (D^2 * 4d^2)^(1/6)*D
    ^(1/6)")
// so the inductance per phase is,

printf(" L = 2 * 10^-4 * log((2^(1/6)*(D^2+d^2)
    ^(1/6) * d^(1/2))/(r^(1/2) * (D^2 + 4d^2)^(1/6)))
    H/km" )
```

Scilab code Exa 4.7 Example

```

// Variable Declaration
r = 0.6         //Radius of each conductor(cm)
d = 150         //Separation distance(cm)
L = 40*10**3    //Length of overhead line(m)
f = 50          //Frequency(Hertz)
v = 50*10**3    //System voltage(V)

// Calculation Section
C_ab = (%pi * 8.854 * 10**(-12))/(log(d/r)) * L    //Capacitance b/w conductors(F)
I = complex(0,v * 2 * %pi * f * C_ab)                    //Charging current leads voltage by 90 (A)

// Result Section
printf('Capacitance between two conductors , C_ab = %.3e F' ,C_ab)
printf('Charging current , I = j%.3f A' ,imag(I))
```

Scilab code Exa 4.8 Example

```

// Variable Declaration
r = 0.015        //Radius of each conductor(m)
D_a1a2 = 0.3     //Distance b/w conductor a1 & a2(m)
D_a2a1 = 0.3     //Distance b/w conductor a2 & a1(m)
D_a1b1 = 15.3    //Distance b/w conductor a1 & b1(m)
D_a1b2 = 15.6    //Distance b/w conductor a1 & b2(m)
D_a2b1 = 15.0    //Distance b/w conductor a2 & b1(m)
```

```
D_a2b2 = 15.3   //Distance b/w conductor a2 & b2(m)
D_b1c1 = 15.3   //Distance b/w conductor b1 & c1(m)
D_b1c2 = 15.6   //Distance b/w conductor b1 & c2(m)
D_b2c1 = 15.0   //Distance b/w conductor b2 & c1(m)
D_b2c2 = 15.3   //Distance b/w conductor b2 & c2(m)
D_a1c1 = 30.6   //Distance b/w conductor a1 & c1(m)
D_a1c2 = 30.9   //Distance b/w conductor a1 & c2(m)
D_a2c1 = 30.3   //Distance b/w conductor a2 & c1(m)
D_a2c2 = 30.6   //Distance b/w conductor a2 & c2(m)

// Calculation Section
D_s = (D_a1a2 * r * D_a2a1 * r)**(1.0/4)                    //Geometric mean radius(m)
D_ab = (D_a1b1 * D_a1b2 * D_a2b1 * D_a2b2)**(1.0/4)         //Mutual GMD b/w conductor a & b(m)
D_bc = (D_b1c1 * D_b1c2 * D_b2c1 * D_b2c2)**(1.0/4)         //Mutual GMD b/w conductor b & c(m)
D_ca = (D_a1c1 * D_a1c2 * D_a2c1 * D_a2c2)**(1.0/4)         //Mutual GMD b/w conductor c & a(m)
D_m = (D_ab * D_bc * D_ca)**(1.0/3)                         //Geometric mean separation(m)
C_n = 2 * %pi * 8.854 * 10**(-9) /(log(D_m/D_s)) //Capacitance per phase(F/km)

// Result Section
printf('Capacitance per phase , C_n = %.3e F/km' , C_n)
```

Scilab code Exa 4.9 Example

```

// Variable Declaration
r = 0.015     //Radius of each conductor(m)
D_ab = 15     //Horizontal distance b/w conductor a &
```

```
b(m)
D_bc = 15     //Horizontal distance b/w conductor b & c(m)
D_ac = 30     //Horizontal distance b/w conductor a & c(m)

// Calculation Section
D_m = (D_ab * D_bc * D_ac)**(1.0/3)                    //Geometric mean separation (m)
D_s = 2**(1.0/2) * r                                        //Geometric mean radius(m)
C_n = 2 * %pi * 8.854 * 10**(-9) /(log(D_m/D_s)) //Capacitance/phase/km(F/km)

// Result Section
printf('Capacitance per phase , C_n = %.3e F/km' , C_n)
```

Scilab code Exa 4.10 Example

```


// calculation of GMD , Dm
// Dab = (da1b1 * da1b2 * da2b1 * da2b2)*(1/4) = (gkkg)^(1/2) = sqrt(gk)
// Inductance/phase  = 2 * 10^-7 log ( Dm / Ds)

printf("Inductance/phase = 2 * 10^-7 / 3 * log(g^2*k^2*h*d/(r^3*f^2*m)) H/m")

// Capacitance/phase = 2*%pi*%e/( log(Dm/Ds))

disp("Capacitance/phase = 6*%pi*%e / (log(g^2*k^2*h*
```

```
d/(r^3*f^2*m))) F/m")
```

Scilab code Exa 4.11 Example

```

// Variable Declaration
h = 5          //Height of conductor above ground(m)
d = 1.5        //Conductor spacing(m)
r = 0.006      //Radius of conductor(m)

// Calculation Section
C_AB = %pi * 8.854*10**-9/log(d/(r*(1+((d*d)/(4*h*h)
    ))**0.5))  //Capacitance with effect of earth(F/
    km)
C_AB1 = %pi * 8.854*10**-9/log(d/r)
                                        //Capacitance ignoring
     effect of earth(F/km)
ch = (C_AB - C_AB1)/C_AB * 100
                                                      //Change
     in capacitance with effect of earth(%)


// Result Section
printf('Line capacitance with effect of earth , C_AB
     = %.3e F/km' ,C_AB)
printf('Line capacitance ignoring effect of earth ,
    C_AB = %.3e F/km' ,C_AB1)
printf('With effect of earth slight increase in
    capacitance = %.1f percent' ,ch)
```

Scilab code Exa 4.12 Example

```
// Variable Declaration
R = 0.16                  //Resistance(ohm)
L = 1.26*10**(-3)         //Inductance(H)
C = 8.77*10**(-9)         //Capacitance(F)
l = 200.0                 //Length of line(km)
P = 50.0                  //Power(MVA)
pf = 0.8                  //Lagging power factor
V_r = 132000.0            //Receiving end voltage(V)
f = 50.0                  //Frequency(Hz)

// Calculation Section
w = 2 * %pi * f
z = complex(R, w*L)          //Series impedance per phase per km(ohm)
y = complex(0, w*C)          //Shunt admittance per phase per km(mho)

g = (y*z)**(0.5)             //propagation constant(/km)
gl = g * l
Z_c = (z/y)**(0.5)           //Surge impedance(ohm)

cosh_gl = cosh(gl)
sinh_gl = sinh(gl)

A = cosh_gl
B = Z_c * sinh_gl
C = (sinh_gl/Z_c)
D = cosh_gl

fi = acos(pf)                                        //Power factor angle(radians)
V_R = V_r/(3**0.5)                                   //Receiving end voltage(V)
I_R = (P*10**6/((3**0.5)*V_r))*(pf - complex(0,sin(fi)))//Receiving end current(A)
```

```
V_S = (A*V_R + B*I_R)                                        //Sending end voltage(V/phase)
V_S_L = V_S * (3**0.5)*10**-3                                //Sending end line voltage(kV)
I_S = C*V_R + D*I_R                                          //Sending end current(A)
pf_S = cos((phasemag(I_S)*%pi/180) - (phasemag(V_S)*%pi/180))          //Sending end power factor
P_S = abs(V_S*I_S)*pf_S*10**-6                               //Sending end power/phase(MW)
P_R = (P/3)*pf                                               //Receiving end power/phase(MW)
P_L = 3*(P_S - P_R)                                          //Total line loss(MW)


// Result Section
printf('Sending end voltage , V_S = %.2f  %.2f  kV/phase',abs(V_S*10**-3),phasemag(V_S))
printf('Sending end line voltage = %.2f kV',abs(V_S_L))
printf('Sending end current , I_S = %.2f  %.2f  A',abs(I_S),phasemag(I_S))
printf('Sending end power factor = %.2f lagging',pf_S)
printf('Total transmission line loss = %.3f MW',P_L)
printf('NOTE : Answers are slightly different because of rounding error.')
```

Scilab code Exa 4.13 Example

```

// Variable Declaration
R = 0.1        //Resistance/phase/km(ohm)
D_m = 800.0    //Spacing b/w conductors(cm)
d = 1.5        //Diameter of each conductor(cm)
l = 300.0      //Length of transmission line(km)
f = 50.0       //Frequency(Hz)

// Calculation Section
L = 2*10**(-4)*log(D_m*2/d)                  //Inductance/phase/km(H)
C = 2*%pi*8.854*10**(-9)/log(D_m*2/d)   //Capacitance/phase/km(F)
w = 2 * %pi * f
z = complex(R, w*L)                          //Series impedance per phase per km(ohm/km)
y = complex(0, w*C)                          //Shunt admittance per phase per km(mho/km)
g = (y*z)**(0.5)                             //propagation constant(/km)
gl = g * l
Z_c = (z/y)**(0.5)                           //Surge impedance(ohm)
sinh_gl = sinh(gl)
tanh_gl = tanh(gl/2)
Z_S = Z_c * sinh_gl                          //Series impedance(ohm)
Y_P = (1/Z_c)*tanh(gl/2)                //Pillar admittance(mho)

// Result Section
printf('Values of equivalent-pi network are :')
```

```
printf('Series impedance , Z_S = (%.2f + j%.2f) ohm'
    ,real(Z_S),imag(Z_S))
printf('Pillar admittance , Y_P = %.2e  % .2f   mho
    = j%.2e mho' ,abs(Y_P),phasemag(Y_P),imag(Y_P))
printf('NOTE : Answers are slightly different
    because of rounding error.')
```

Scilab code Exa 4.14 Example

```

// Variable Declaration
V_r = 220000.0    //Voltage(V)
P = 100.0         //Power(MW)
r = 0.08          //Series resistance(ohm)
x = 0.8           //Series reactance(ohm)
s = 6.0*10**(-6)  //Shunt susceptance(mho)
pf = 0.8          //Power factor lagging
l_1 = 60.0        //Transmission length(km) for case(
    i)
l_2 = 200.0       //Transmission length(km) for case(
    ii)
l_3 = 300.0       //Transmission length(km) for case(
    iii)
l_4 = 500.0       //Transmission length(km) for case(
    iv)

// Calculation Section
z = complex(r,x)

    //Series impedance/km(ohm)
y = complex(0,s)

    //Shunt admittance/km(mho)
theta_R = acos(pf)
P_R = P/3
```

```
//Active power at receiving end/phase(MW)
Q_R = (P/3)*tan(theta_R)                                    //Reactive power at receiving end/phase(MVAR)
V_R = V_r/(3**0.5)                                          //Receiving end voltage/phase(V)
I_R = P*10**6/((3**0.5)*V_r*pf)*(pf - complex(0,sin(theta_R)))//Receiving end current(A)
Z_c = (z/y)**(0.5)                                          //Surge impedance(ohm)

A_1 = 1                                                     //Constant A
B_1 = z*l_1                                                 //Constant B(ohm)
C_1 = 0                                                     //Constant C(mho)
D_1 = A_1                                                   //Constant D
V_S_1 = A_1*V_R + B_1*I_R                                   //Sending end voltage(V/phase)
I_S_1 = I_R                                                 //Sending end current(A)
theta_S_1 = (phasemag(I_S_1)*%pi/180) - (phasemag(V_S_1)*%pi/180)     //Sending end power factor
P_S_1 = abs(V_S_1*I_S_1)*cos(theta_S_1)*10**-6              //Sending end power(MW)
n_1 = (P_R/P_S_1)*100                                       //Transmission
```

```
    efficiency(%)
reg_1 = (abs(V_S_1/A_1) - V_R)/V_R*100                  //Regulation(%)
Q_S_1 = V_S_1 * conj(I_S_1)*10**-6                      //Sending end reactive power(MVAR)
Q_line_1 = imag(Q_S_1) - Q_R                            //Reactive power absorbed by line(MVAR)

Z_S_2 = z*l_2
Y_P_2 = y*l_2/2
A_2 = 1 + Y_P_2*Z_S_2
B_2 = Z_S_2
C_2 = Y_P_2*(2 + Y_P_2*Z_S_2)
D_2 = A_2
V_S_2 = A_2*V_R + B_2*I_R                   //Sending end voltage(V/phase)
I_S_2 = C_2*V_R + D_2*I_R                   //Sending end current(A)
S_S_2 = V_S_2*conj(I_S_2)*10**-6 //Sending end complex power(MVA)
P_S_2 = real(S_S_2)                          //Power at sending end(MW)
n_2 = (P_R/P_S_2)*100                       //Transmission efficiency(%)
reg_2 = (abs(V_S_2/A_2) - V_R)/V_R*100 //Regulation(%)
Q_line_2 = imag(S_S_2) - Q_R                 //Reactive power absorbed by line(MVAR)

g_3 = (y*z)**(0.5)                          //propagation constant(/km)
gl_3 = g_3 * l_3
cosh_gl_3 = cosh(gl_3)
sinh_gl_3 = sinh(gl_3)
A_3 = cosh_gl_3
B_3 = Z_c * sinh_gl_3
C_3 = sinh_gl_3/Z_c
```

```
D_3 = cosh_gl_3
V_S_3 = A_3*V_R + B_3*I_R                    //Sending end voltage(V/phase)
I_S_3 = C_3*V_R + D_3*I_R                    //Sending end current(A)
S_S_3 = V_S_3*conj(I_S_3)*10**-6 //Sending end complex power(MVA)
P_S_3 = real(S_S_3)                           //Power at sending end(MW)
n_3 = (P_R/P_S_3)*100                        //Transmission efficiency(%)
reg_3 = (abs(V_S_3/A_3) - V_R)/V_R*100 //Regulation(%)
Q_line_3 = imag(S_S_3) - Q_R                  //Reactive power absorbed by line(MVAR)

g_4 = (y*z)**(0.5)                           //propagation constant(/km)
gl_4 = g_4 * l_4
cosh_gl_4 = cosh(gl_4)
sinh_gl_4 = sinh(gl_4)
A_4 = cosh_gl_4
B_4 = Z_c * sinh_gl_4
C_4 = sinh_gl_4/Z_c
D_4 = cosh_gl_4
V_S_4 = A_4*V_R + B_4*I_R                    //Sending end voltage(V/phase)
I_S_4 = C_4*V_R + D_4*I_R                    //Sending end current(A)
S_S_4 = V_S_4*conj(I_S_4)*10**-6 //Sending end complex power(MVA)
P_S_4 = real(S_S_4)                           //Power at sending end(MW)
n_4 = (P_R/P_S_4)*100                        //Transmission efficiency(%)
reg_4 = (abs(V_S_4/A_4) - V_R)/V_R*100 //Regulation(%)
Q_line_4 = imag(S_S_4) - Q_R                  //Reactive
```

```
    power absorbed by line(MVAR)

// Result Section
printf('Case(i) : For Length = 60 km')
printf('Efficiency , n = %.2f percent',n_1)
printf('Regulation = %.3f percent',reg_1)
printf('Reactive power at sending end , Q_S = %.2f
    MVAR',imag(Q_S_1))
printf('Reactive power absorbed by line , Q_line = %
    .2f MVAR',Q_line_1)
printf('\nCase(ii) : For Length = 200 km')
printf('Efficiency , n = %.2f percent',n_2)
printf('Regulation = %.2f percent',reg_2)
printf('Reactive power at sending end , Q_S = %.2f
    MVAR',imag(S_S_2))
printf('Reactive power absorbed by line , Q_line = %
    .2f MVAR',Q_line_2)
printf('\nCase(iii) : For Length = 300 km')
printf('Efficiency , n = %.2f percent',n_3)
printf('Regulation = %.2f percent',reg_3)
printf('Reactive power at sending end , Q_S = %.2f
    MVAR',imag(S_S_3))
printf('Reactive power absorbed by line , Q_line = %
    .2f MVAR',Q_line_3)
printf('\nCase(iv) : For Length = 500 km')
printf('Efficiency , n = %.2f percent',n_4)
printf('Regulation = %.2f percent',reg_4)
printf('Reactive power at sending end , Q_S = %.2f
    MVAR',imag(S_S_4))
printf('Reactive power absorbed by line , Q_line = %
    .2f MVAR',Q_line_4)
printf('\nNOTE : ERROR : Calculation mistake in case
    (iv) efficiency in textbook')
```

Scilab code Exa 4.16 Example

```
// Variable Declaration
A = 0.8*exp(%i*1.4*%pi/180)      //Line constant
B = 326.0*exp(%i*84.8*%pi/180)   //Line constant(ohm)
V_R = 220.0                                //Receiving end voltage(kV)
V_S = 220.0                                //Sending end voltage(kV)
P = 75.0                                   //Power(MVA) for case(a)
pf = 0.8                                   //Power factor lagging

a = phasemag(A)*%pi/180                    //Phase angle of A(radian)
b = phasemag(B)*%pi/180                    //Phase angle of B(radian)

// Calculation Section
P_R = P * pf                               //Active power demanded by load(MW)
P_React = P *(1-pf**2)**0.5                //Reactive power demanded by load(MVAR)
cos_b_delta_1 = P_R*abs(B)/(V_R*V_S) + abs(A)*cos(b-a)          //cos(b-delta)[in radians]
delta_1 = b - acos(cos_b_delta_1)                               //delta(radians)
Q_R_1 = (V_R*V_S/abs(B))*sin(b-delta_1) - (abs(A)*V_R**2/abs(B))*sin(b-a) //Reactive power at sending end(MVAR)
Reactive_power_1 = P_React - Q_R_1         //Reactive power to be supplied by compensating equipment(MVAR)
```

```
cos_b_delta_2 = (abs(A)*V_R/V_S)*cos(b-a)    //cos(b-delta)[in radians]
delta_2 = b - acos(cos_b_delta_2)    //delta(radians)
Q_R_2 = (V_R*V_S/abs(B))*sin(b-delta_2) - (abs(A)*V_R**2/abs(B))*sin(b-a) //Reactive power at sending end(MVAR)
Reactive_power_2 = Q_R_2    //Reactive power to be absorbed by compensating equipment(MVAR)

// Result Section
printf('(a) Reactive VARs to be supplied by compensating equipment = %.2f MVAR' , Reactive_power_1)
printf('(b) Reactive VARs to be absorbed by compensating equipment = %.2f MVAR' , Reactive_power_2)
```

Scilab code Exa 4.17 Example

```

// Variable Declaration
r = 25.0        //Resistance/phase(ohm)
x = 90.0        //Reactance/phase(ohm)
V_S = 145.0     //Sending end voltage(kV)
V_R = 132.0     //Receiving end voltage(kV)
P_R_1 = 0       //Power(MW)
P_R_2 = 50.0    //Power(MW)

// Calculation Section
```

```
A = 1.0*exp(%i*0*%pi/180)          //Line constant
B = complex(r,x)                   //Line constant(ohm)
a = phasemag(A)*%pi/180            //Phase angle of A(radian)
b = phasemag(B)*%pi/180            //Phase angle of B(radian)

cos_b_delta_1 = (V_R/V_S)*cos(b-a)
delta_1 = b - acos(cos_b_delta_1)
Q_R_1 = (V_R*V_S/abs(B))*sin(b-delta_1) - (abs(A)*V_R**2/abs(B))*sin(b-a)

cos_b_delta_2 = (P_R_2*abs(B)/(V_R*V_S))+(abs(A)*V_R/V_S)*cos(b-a)
delta_2 = (b - acos(cos_b_delta_2))
Q_R_2 = (V_R*V_S/abs(B))*sin(b-delta_2)-(abs(A)*V_R**2/abs(B))*sin(b-a) //Reactive power available at receiving end(MVAR)
Q_S_2 = Q_R_1 + Q_R_2              //Reactive power to be supplied by equipment(MVAR)
pf = cos(atan(Q_S_2/P_R_2))        //Power factor

// Result Section
printf('Rating of device = %.2f MVAR',Q_R_1)
printf('Power factor = %.2f lagging',pf)
```

Scilab code Exa 4.18 Example

```

// Variable Declaration
```

```
A = 0.9*exp(%i*1.0*%pi/180)      //Line constant
B = 143.0*exp(%i*84.5*%pi/180)   //Line constant(ohm)
V_R = 220.0                                    //Receiving end voltage(kV)
V_S = 240.0                                    //Sending end voltage(kV)
P = 100.0                                      //Power(MVA)
pf = 0.8                                       //Power factor lagging

a = phasemag(A)*%pi/180                                  //Phase angle of A(radian)
b = phasemag(B)*%pi/180                                  //Phase angle of B(radian)

// Calculation Section
P_R = P * pf                                   //Active power at receiving end(MW)
cos_b_delta = (P_R*abs(B)/(V_R*V_S))+(abs(A)*V_R/V_S)*cos(b-a)          //cos(b-delta)[in radians]
delta_1 = (b - acos(cos_b_delta))
Q_R = (V_R*V_S/abs(B))*sin(b-delta_1)-(abs(A)*V_R**2/abs(B))*sin(b-a) //Reactive power at receiving end(MVAR)
P_Re = P *(1-pf**2)**0.5                       //Reactive power(MVAR)
rating = P_Re - Q_R                            //Rating of phase modifier(MVAR)

delta_2 = b                                    //Maximum power is received when delta = b
P_Rmax = (V_R*V_S/abs(B))-(abs(A)*V_R**2/abs(B))*cos(b-a)              //Maximum power at
```

```
receiving end(MW)
Q_R = -(abs(A/B)*V_R**2)*sin(b-a)                                    //Reactive power at receive end(MVAR)
P_S = (V_S**2*abs(A/B))*cos(b-a)-(V_S*V_R/abs(B))*cos(b+delta_2)      //Sending end power(MW)
n_line = (P_Rmax/P_S)*100                                            //Line efficiency(%)

// Result Section
printf('Case(a) :')
printf('Rating of phase modifier = %.3f MVAR' ,rating)
printf('Power angle , delta = %.2f ° ',(delta_1*180/%pi))
printf('\nCase(b) :')
printf('Maximum power at receive end , P_Rmax = %.2f MW' ,P_Rmax)
printf('Reactive power available , Q_R = %.2f MVAR' ,Q_R)
printf('Line efficiency = %.2f percent' ,n_line)
```

Scilab code Exa 4.19 Example

```

// Variable Declaration
A = 0.96*exp(%i*1.0*%pi/180)     //Line constant
B = 100.0*exp(%i*83.0*%pi/180)   //Line constant(ohm)
V_R = 110.0                      //Receiving end voltage(kV)
V_S = 110.0                      //Sending end voltage(kV)
pf = 0.8                         //Power factor lagging
```

```
delta = 15*%pi/180                    //Power angle(radians)

// Calculation Section
a = phasemag(A)*%pi/180               //Phase angle of A(radian)
b = phasemag(B)*%pi/180               //Phase angle of B(radian)

P_R = (V_R*V_S/abs(B))*cos(b-delta) - (abs(A/B)*V_R**2)*cos(b-a) //Active power at receiving end(MW)
Q_RL = P_R*tan(acos(pf))              //Reactive power demanded by load(MVAR)

Q_R = (V_R*V_S/abs(B))*sin(b-delta) - (abs(A/B)*V_R**2)*sin(b-a) //Reactive power(MVAR)
rating = Q_RL - Q_R                   //Rating of device(MVAR)

P_S = (V_S**2*abs(A/B))*cos(b-a) - (V_R*V_S/abs(B))*cos(b+delta) //Sending end active power(MW)
n_line = (P_R/P_S)*100                //Efficiency of line(%)

Q_S = (V_S**2*abs(A/B))*sin(b-a) - (V_R*V_S/abs(B))*sin(b+delta) //Sending end reactive power(MVAR)

// Result Section
printf('(i)  Active power demanded by load , P_R = %.2f MW',P_R)
printf('     Reactive power demanded by load , Q_RL = %.2f MVAR',Q_RL)
printf('(ii) Rating of the device , Q_R = %.2f MVAR',rating)
printf('(iii)Efficiency of line = %.2f percent',
```

```
    n_line)
printf('(iv) Reactive power supplied by source and
    line , Q_S = %.2f MVAR' ,Q_S)
```

Chapter 5

OVERHEAD LINE CONSTRUCTION

Scilab code Exa 5.1 Example

```
// Variable Declaration
L = 250.0            //Span(m)
d = 1.1*10**-2       //Conductor diameter(m)
w = 0.650*9.81       //Conductor weight(N/m)
bl = 7000.0          //Breaking load(kg)
sf = 2               //Safety factor
P_w_2 = 350.0        //Wind pressure(N/m^2) for case(ii)
P_w_3 = 400.0        //Wind pressure(N/m^2) for case(iii)
t_3 = 10.0**-2       //Thickness of ice covering(m) for case(iii)
w_ice = 915.0        //Ice weight(kg/m^3)

// Calculation Section
T_0 = (bl/sf)*9.81    //Allowable tension(N)

S_1 = (T_0/w)*(cosh(w*L/(2*T_0))-1)          //Sag(m)
```

```
S_1_1 = (w*L**2)/(8*T_0)                          //Sag using parabolic equation(m)

F_w_2 = P_w_2 * d                                 //Wind force(N/m)
w_t_2 = (w**2 + F_w_2**2)**0.5                    //Total force on conductor(N/m)
S_2 = (T_0/w_t_2)*(cosh(w_t_2*L/(2*T_0))-1)   //Sag(m)
S_2_2 = w_t_2*L**2/(8*T_0)                        //Sag using parabolic equation(m)
alpha_2 = atan(F_w_2/w)                       //w_t inclined vertical angle(radians)
S_v_2 = S_2 * cos(alpha_2)                    //Vertical component of sag(m)

D_3 = d + 2*t_3                                   //Diameter of conductor with ice(m)
F_w_3 = P_w_3 * D_3                               //Wind force(N/m)
w_ice_3 = (%pi/4)*(D_3**2 - d**2)*w_ice*9.81   //Weight of ice(N/m)
w_t_3 = ((w+w_ice_3)**2 + F_w_3**2)**0.5          //Total force on conductor(N/m)
S_3 = (T_0/w_t_3)*(cosh(w_t_3*L/(2*T_0))-1)   //Sag(m)
S_3_3 = w_t_3*L**2/(8*T_0)                        //Sag using parabolic equation(m)
alpha_3 = atan(F_w_3/(w+w_ice_3))             //w_t inclined vertical angle(radians)
S_v_3 = S_3 * cos(alpha_3)                    //Vertical component of sag(m)

// Result Section
printf('Case(i) :')
printf('Sag using catenary equation = %.4f m ',S_1)
printf('Sag using parabolic equation = %.4f m \n',S_1_1)
```

```
printf('Case(ii) :')
printf('Sag using catenary equation = %.4f m ',S_2)
printf('Sag using parabolic equation = %.4f m ',
   S_2_2)
printf('Vertical component of sag = %.2f m \n',
   S_v_2)
printf('Case(iii) :')
printf('Sag using catenary equation = %.4f m ',S_3)
printf('Sag using parabolic equation = %.4f m ',
   S_3_3)
printf('Vertical component of sag = %.3f m \n',
   S_v_3)
```

Scilab code Exa 5.2 Example

```
// Variable Declaration
w = 0.85                //Weight of overhead line(kg/m)
T_0 = 3.5*10**4         //Maximum allowable tension(N)
L_1 = 160.0             //Span(m) for case(i)
L_2 = 200.0             //Span(m) for case(ii)
L_3 = 250.0             //Span(m) for case(iii)
L_4 = 275.0             //Span(m) for case(iv)
g_c = 7.1               //Minimum ground clearance(m)
L_S = 1.5               //Length of suspension
   insulator string

// Calculation Section
w1 = w * 9.81               //Weight(N/m)

S_1 = w1*L_1**2/(8*T_0)    //Sag(m)
H_1 = g_c + S_1 + L_S      //Height of lowest cross-
   arm(m)

S_2 = w1*L_2**2/(8*T_0)    //Sag(m)
H_2 = g_c + S_2 + L_S      //Height of lowest cross-
```

```
    arm(m)

S_3 = w1*L_3**2/(8*T_0)   //Sag(m)
H_3 = g_c + S_3 + L_S     //Height of lowest cross-
    arm(m)

S_4 = w1*L_4**2/(8*T_0)   //Sag(m)
H_4 = g_c + S_4 + L_S     //Height of lowest cross-
    arm(m)

// Result Section
printf('Span in meters\t                         %d
    \t  %d\t  %d\t  %d',L_1,L_2,L_3,L_4)
printf('Sag in meters\t                        %.3f
    \t %.3f\t %.3f\t %.3f',S_1,S_2,S_3,S_4)
printf('Height of lowest cross-arm in meters\t %.3f\
    t %.3f\t %.3f\t %.3f',H_1,H_2,H_3,H_4)
printf('\nNOTE : ERROR : For finding height of
    lowest cross arm the length of insulation string
    is not considered in textbook calculation')
printf('although it is mentioned in formula. Since
    length of insulation string is taken here there
    is a difference in answers from that of given in
    textbook')
```

Scilab code Exa 5.3 Example

```

// Variable Declaration
w = 0.63            //Weight of conductor(kg/m)
T_0 = 1350.0        //Maximum allowable load(kg)
h_1 = 20.0          //Height of first tower(m)
h_2 = 15.0          //Height of second tower(m)
L = 240.0           //Span(m)

```

```
// Calculation Section
h = h_1 - h_2                  //Difference in levels of towers(m)
L_1 = (L/2)+(T_0*h/(w*L))      //Horizontal distance from higher support(m)
L_2 = (L/2)-(T_0*h/(w*L))      //Horizontal distance from lower support(m)
S_1 = w*L_1**2/(2*T_0)         //Sag from upper support(m)
S_2 = w*L_2**2/(2*T_0)         //Sag from lower support(m)
clearance = (h_1 - S_1)        //Minimum clearance(m)

// Result Section
printf('Minimum clearance between a line conductor & water surface = %.3f m' ,clearance)
printf('Sag from upper support = %.3f m' ,S_1)
```

Scilab code Exa 5.5 Example

```

// Variable Declaration
n = 3        //Number of discs
m = 0.1      //capacitance of each link pin to self capacitance
V = 33.0     //Voltage(kV)

// Calculation Section
a_1 = 1
a_2 = (1 + m)*a_1
a_3 = m*(a_1 + a_2) + a_2
v_1 = V/(a_1 + a_2 + a_3)     //Voltage across top unit(kV)
v_2 = a_2 * v_1               //Voltage across middle unit(kV)
```

```
v_3 = a_3 * v_1                //Voltage across bottom unit(kV)
s_v_1 = (v_1/V)*100            //Voltage across top unit to string voltage(%)
s_v_2 = (v_2/V)*100            //Voltage across middle unit to string voltage(%)
s_v_3 = (v_3/V)*100            //Voltage across bottom unit to string voltage(%)

efficiency = V*100/(3*v_3)   //String efficiency(%)

// Result Section
printf('Case(i) :')
printf('Voltage across top unit , v_1 = %.3f kV' ,v_1)
printf('Voltage across middle unit , v_2 = %.3f kV',v_2)
printf('Voltage across bottom unit , v_3 = %.3f kV',v_3)
printf('Voltage across top unit as a percentage of string voltage , v_1/V = %.1f percent' ,s_v_1)
printf('Voltage across middle unit as a percentage of string voltage , v_2/V = %.1f percent' ,s_v_2)
printf('Voltage across bottom unit as a percentage of string voltage , v_3/V = %.1f percent' ,s_v_3)
printf('\nCase(ii) :')
printf('String efficiency = %.2f percent' , efficiency)
```

Scilab code Exa 5.6 Example

```
// Variable Declaration
n = 8          //Number of discs
m = 1.0/6      //capacitance of each link pin to self capacitance
```

```
V = 30.0    //Voltage(kV)

// Calculation Section
a_1 = 1
a_2 = (1+m)*a_1
a_3 = m*(a_1+a_2)+a_2
a_4 = m*(a_1+a_2+a_3)+a_3
a_5 = m*(a_1+a_2+a_3+a_4)+a_4
a_6 = m*(a_1+a_2+a_3+a_4+a_5)+a_5
a_7 = m*(a_1+a_2+a_3+a_4+a_5+a_6)+a_6
a_8 = m*(a_1+a_2+a_3+a_4+a_5+a_6+a_7)+a_7
v_1 = V/(a_1+a_2+a_3+a_4+a_5+a_6+a_7+a_8)    //Voltage across unit 1(kV)
v_2 = a_2*v_1    //Voltage across unit 2(kV)
v_3 = a_3*v_1    //Voltage across unit 3(kV)
v_4 = a_4*v_1    //Voltage across unit 4(kV)
v_5 = a_5*v_1    //Voltage across unit 5(kV)
v_6 = a_6*v_1    //Voltage across unit 6(kV)
v_7 = a_7*v_1    //Voltage across unit 7(kV)
v_8 = a_8*v_1    //Voltage across unit 8(kV)
s_v_1 = v_1/V*100    //Voltage across unit 1 as a % of V
s_v_2 = v_2/V*100    //Voltage across unit 2 as a % of V
s_v_3 = v_3/V*100    //Voltage across unit 3 as a % of V
s_v_4 = v_4/V*100    //Voltage across unit 4 as a % of V
s_v_5 = v_5/V*100    //Voltage across unit 5 as a % of V
s_v_6 = v_6/V*100    //
```

```
Voltage across unit 6 as a % of V
s_v_7 = v_7/V*100                                  //Voltage across unit 7 as a % of V
s_v_8 = v_8/V*100                                  //Voltage across unit 8 as a % of V

V_2 = V*100/s_v_8
V_sys = (3**0.5)*V_2                               //Permissible system voltage(kV)

// Result Section
printf('Case(i) :')
printf('Unit number                          1    2    3    4    5    6    7    8\n')
printf('Percentage of conductor voltage   %.2f   %.2f   %.2f   %.2f   %.2f   %.2f   %.2f   %.2f',s_v_1,s_v_2,s_v_3,s_v_4,s_v_5,s_v_6,s_v_7,s_v_8)
printf('\nCase(ii) :')
printf('System voltage at which this string can be used = %.2f kV',V_sys)
```

Scilab code Exa 5.7 Example

```

// Variable Declaration
v_dry = 65.0      //Dry power frequency flashover voltage for each disc(kV)
v_wet = 43.0      //Wet power frequency flashover voltage for each disc(kV)
V = 110           //Voltage of system to be insulated(kV)
m = 1.0/6         //capacitance of each link pin to self capacitance
n_4 = 4           //Number of units in a string
```

```
n_8 = 8            //Number of units in a string
n_10 = 10          //Number of units in a string
v_dry_4 = 210.0   //Dry power frequency flashover voltage for 4 units(kV)
v_dry_8 = 385.0   //Dry power frequency flashover voltage for 8 units(kV)
v_dry_10 = 460.0 //Dry power frequency flashover voltage for 10 units(kV)
v_wet_4 = 150.0   //Wet power frequency flashover voltage for 4 units(kV)
v_wet_8 = 285.0   //Wet power frequency flashover voltage for 8 units(kV)
v_wet_10 = 345.0 //Wet power frequency flashover voltage for 10 units(kV)

// Calculation Section
eff_dry_4 = v_dry_4*100/(n_4*v_dry)
eff_dry_8 = v_dry_8*100/(n_8*v_dry)
eff_dry_10 = v_dry_10*100/(n_10*v_dry)
eff_wet_4 = v_wet_4*100/(n_4*v_wet)
eff_wet_8 = v_wet_8*100/(n_8*v_wet)
eff_wet_10 = v_wet_10*100/(n_10*v_wet)

a_1 = 1
a_2 = (1+m)*a_1
a_3 = m*(a_1+a_2)+a_2
a_4 = m*(a_1+a_2+a_3)+a_3
a_5 = m*(a_1+a_2+a_3+a_4)+a_4
a_6 = m*(a_1+a_2+a_3+a_4+a_5)+a_5
a_7 = m*(a_1+a_2+a_3+a_4+a_5+a_6)+a_6
a_8 = m*(a_1+a_2+a_3+a_4+a_5+a_6+a_7)+a_7
v_1 = V/(a_1+a_2+a_3+a_4+a_5+a_6+a_7+a_8)     //Voltage across unit 1(kV)
v_8 = a_8*v_1                                  //Voltage across unit 8(kV)
s_v_8 = v_8/V                                  //Ratio of Voltage across unit 8 to string voltage
voltage_2 = V/(3**0.5)*s_v_8                   //
```

```
Voltage across the disc adjacent to conductor(kV)
sf_dry = v_dry/voltage_2                            //Factor of safety for dry flashover
sf_wet = v_wet/voltage_2                            //Factor of safety for wet flashover


// Result Section
printf('Case(i) :')
printf(' No. of units        Dry string efficiency(%%)      Wet string efficiency(%%)')

printf(' %d                      %.2f                      %.2f          ',n_4,eff_dry_4,eff_wet_4)
printf(' %d                      %.2f                      %.2f          ',n_8,eff_dry_8,eff_wet_8)
printf(' %d                     %.2f                      %.2f          ',n_10,eff_dry_10,eff_wet_10)

printf('\nCase(ii) :')
printf('Factor of safety for dry flashover = %.2f' ,sf_dry)
printf('Factor of safety for wet flashover = %.2f' ,sf_wet)
```

Scilab code Exa 5.8 Example

```
// Variable Declaration
n = 4          //Number of disc
v_2 = 13.2     //Voltage across second unit(kV)
v_3 = 18.0     //Voltage across third unit(kV)

```

```
// Calculation Section
m = 0.198                              //Obtained by solving the quadratic equation
a_1 = 1
a_2 = 1+m
a_3 = m*(a_1+a_2)+a_2
a_4 = m*(a_1+a_2+a_3)+a_3
v_1 = v_2/a_2                          //Voltage across first unit(kV)
v_4 = a_4*v_1                          //Voltage across second unit(kV)
V = v_1+v_2+v_3+v_4                    //Conductor voltage(kV)
efficiency = V/(n*v_4)*100             //String efficiency(%)

// Result Section
printf('Conductor voltage with respect to the cross-arm , V = %.2f kV' ,V)
printf('String efficiency = %.2f percent' , efficiency)
```

Scilab code Exa 5.9 Example

```

// Variable Declaration
n = 3        //Number of disc

unit_1 = 100/3.072                     //Disc voltage as % of conductor voltage of Topmost unit
unit_2 = 1.014/3.072*100               //Disc voltage as % of conductor voltage of second unit
unit_3 = 1.058/3.072*100               //Disc voltage as % of conductor voltage of bottom unit
efficiency = 3.072*100/(n*1.058)       //String
```

```
efficiency(%)

// Calculation Section
unit_1g = 100/3.752                     //Disc voltage as % of conductor voltage of Topmost unit
unit_2g = 1.18/3.752*100                //Disc voltage as % of conductor voltage of second unit
unit_3g = 1.5724/3.752*100              //Disc voltage as % of conductor voltage of bottom unit
efficiency1 = 3.752*100/(n*1.5724)      //String efficiency(%)

// Result Section
printf('Disc voltages as a percentage of the conductor voltage with guard ring are :')
printf('Topmost unit = %.2f percent',unit_1)
printf('Second unit = %.2f percent',unit_2)
printf('Bottom unit = %.2f percent',unit_3)
printf('String efficiency = %.2f percent',efficiency)
printf('\nDisc voltages as a percentage of the conductor voltage without guard ring are :')
printf('Topmost unit = %.2f percent',unit_1g)
printf('Second unit = %.2f percent',unit_2g)
printf('Bottom unit = %.2f percent',unit_3g)
printf('String efficiency = %.2f percent',efficiency1)
```

Scilab code Exa 5.10 Example

```

// Variable Declaration
v = 220.0       //Voltage(kV)
f = 50.0        //Frequency(Hertz)
p = 752.0       //Pressure(mm of Hg)
```

```
t = 40.0      //Temperature( C )
m = 0.92      //Surface irregularity factor
r = 1.2       //Conductor radius(cm)
d = 550.0     //Spacing(cm)

// Calculation Section
delta = (0.392*p)/(273+t)             //Air density correction factor
V_c = 21.1*delta*m*r*log(d/r)  //Corona inception voltage(kv/phase)rms
V_c_l = 3**0.5*V_c                    //Line-line corona inception voltage(kV)

// Result Section
printf('Corona inception voltage , V_c = %.2f kV/phase' ,V_c)
printf('Line-to-line corona inception voltage = %.2f kV' ,V_c_l)
```

Scilab code Exa 5.11 Example

```

// Variable Declaration
v = 220.0     //Voltage(kV)
f = 50.0      //Frequency(Hertz)
v_o = 1.6     //Over voltage(p.u)
p = 752.0     //Pressure(mm of Hg)
t = 40.0      //Temperature( C )
m = 0.92      //Surface irregularity factor
r = 1.2       //Conductor radius(cm)
d = 550.0     //Spacing(cm)

// Calculation Section
delta = (0.392*p)/(273+t)
                                      //Air density
```

```
correction factor
V_c = 21.1*delta*m*r*log(d/r)                    //Corona inception voltage(kv/phase)rms
V_ph = (v * v_o)/3**0.5                          //Phase voltage(kV)
peek = 3*(241/delta)*(f+25)*(r/d)**0.5*(V_ph-V_c)**2*10**-5   //Peek's formula(kW/km)
ratio = V_ph/V_c
F = 0.9                                          //Ratio of V_ph to V_c
peterson = 3*2.1*f*F*(V_c/log10(d/r))**2*10**-5        //Peterson's formula(kW/km)

// Result Section
printf('Corona loss using Peeks formula , P = %.2f kW/km' ,peek)
printf('Corona loss using Petersons formula , P = %.2f kW/km' ,peterson)
```

Chapter 6

UNDERGROUND CABLES

Scilab code Exa 6.1 Example

```
// Variable Declaration
C_m = 0.28         //Capacitance b/w ant 2 cores(micro-
    F/km)
f = 50.0           //Frequency(Hz)
V_L = 11.0         //Line voltage(kV)

// Calculation Section
C = 2*C_m                                 //Capacitance b/
    w any conductor & shield(micro-F/km)
w = 2*%pi*f                               //Angular frequency
I_c = V_L*10**3*w*C*10**-6/3**0.5    //Charging
    current/phase/km(A)
Total = 3**0.5*I_c*V_L                    //Total charging
     kVAR/km

// Result Section
printf('Charging current/phase/km , I_c = %.3f A' ,
    I_c)
printf('Total charging kVAR/km = %.2f ' ,Total)
```

Scilab code Exa 6.2 Example

```
// Variable Declaration
E_c = 100.0      //Safe working stress(kV/cm) rms
V = 130.0        //Operating voltage(kV) rms
d = 1.5          //Diameter of conductor(cm)

// Calculation Section
ln_D = 2*V/(E_c*d)+log(d)
D = exp(ln_D)
thick_1 = (D-d)/2                //Insulation thickness(cm)

d_2 = 2*V/E_c
D_2 = 2.718*d_2                  //Sheath diameter(cm)
thick_2 = (D_2-d_2)/2            //Insulation thickness(cm)

// Result Section
printf('(i) Internal sheath radius = %.2f cm' , thick_1)
printf('(ii) Internal sheath radius = %.2f cm' , thick_2)
```

Scilab code Exa 6.3 Example

```
// Variable Declaration
d = 3.0          //Diameter of conductor(cm)
D = 8.5          //Sheath diameter(cm)
e_r1 = 5.0       //Permittivity of inner dielectric
e_r2 = 3.0       //Permittivity of outer dielectric
```

```
E_c = 30.0        //Safe working stress(kV/cm) rms

// Calculation Section
E_i = E_c
D_1 = e_r1/e_r2*d
thick_1 = (D_1-d)/2       //Thickness of first layer(cm)
thick_2 = (D-D_1)/2       //Thickness of second layer(cm)

V_1 = E_c*d*log(D_1/d)/2          //Voltage across first layer(kV)
V_2 = E_i*D_1*log(D/D_1)/2        //Voltage across second layer(kV)
V = V_1 + V_2                     //Permissible conductor voltage(kV)

V_3 = E_c*d*log(D/d)/2            //Permissible conductor voltage(kV) for homogeneous permittivity of 5


// Result Section
printf('Case(i) :')
printf('Thickness of first layer = %.2f cm',thick_1)
printf('Thickness of second layer = %.2f cm',thick_2)
printf('\nCase(ii) :')
printf('Permissible conductor voltage = %.2f kV',V)
printf('\nCase(iii) :')
printf('Permissible conductor voltage if a homogeneous insulation of permittivity 5 is used, V = %.2f kV',V_3)
printf('\nNOTE : ERROR : Relative permittivity of outer dielectric is 3 & not 9 as given in textbook')
```

Scilab code Exa 6.4 Example

```
// Variable Declaration
E = 40.0           //Safe working stress(kV/cm) rms
d = 1.5            //Conductor diameter(cm)
D = 6.7            //Sheath diameter(cm)
t = 0.1            //Thickness of lead tube(cm)


// Calculation Section
r = d/2                          //Conductor radius(cm)
R = D/2                          //Sheath radius(cm)
r_i = r+((R-r)/2)-t/2            //Internal radius of intersheath(cm)
r_e = r_i + t                    //External radius of intersheath(cm)
V_1 = E*r*log(r_i/r)     //Voltage across conductor & intersheath(kV)
V_2 = E*r_e*log(R/r_e)  //Voltage across intersheath & earthed sheath(kV)
V = V_1 + V_2                    //Safe working voltage with intersheath(kV)
V_no = E*r*log(R/r)      //Safe working voltage without intersheath(kV)

// Result Section
printf('Safe working voltage with intersheath , V = %.2f kV' ,V)
printf('Safe working voltage without intersheath , V = %.2f kV' ,V_no)
```

Chapter 7

SUBSTATION AND DISTRIBUTION SYSTEM

Scilab code Exa 7.1 Example

```
//find..
    clc
    //solution
    //given
    Tb=3*10^6//N-mm
    d=1//m
    r=500//mm
    u=0.3
    q=0.61//rad
    ub=(4*u*sin(q))/(2*q+sin(2*q))//eqivalent
       coffint of friction
    //ref fig 25.12
    //let S be spring force
    //taking moment abt fulcrum O1
    //S*1250=Rn1*600 + Ft1*(500-250)
    //put Rn1=Ft1/ub...
    //S*1250=2125*Ft1
    //Ft1=S*1250/2125
    //taking moment abt O2
```

```
    //S*1250+Ft2(500-250)=Rn2*600
    //Rn2=Ft2/ub
    //S*1250+Ft2(500-250)=1625Ft2
    //Ft2=S*1250/1625
    //Tb=(Ft1+Ft2)*r=680*S
    S=Tb/680
    printf("spring force is,%f N\n",S)
    //let b be width of brakes shoes
    //Ab=b*(2*r*sin(q))//mm
    Ft1=S*1250/2125
    Rn1=Ft1/ub
    Ft2=S*1250/1625
    Rn2=Ft2/ub// Variable Declaration
V = 400.0       //Voltage supplied(V)
f = 50.0        //Frequency(Hz)
P_1 = 75.0      //Power of induction motor at middle of distributor(kVA)
pf_1 = 0.8      //Power factor of induction motor at middle of distributor
P_2 = 50.0      //Power of induction motor at far end(kVA)
pf_2 = 0.85     //Power factor of induction motor at far end
demand_f = 1.0  //Demand factor
diver_f = 1.2   //Diversity factor
L = 150.0       //Length of line(m)

// Calculation Section
theta_1 = acos(pf_1)                                        //Power factor angle for 75 kVA(radians)
theta_2 = acos(pf_2)                                        //Power factor angle for 50 kVA(radians)
load = P_1*exp(%i*theta_1)+P_2*exp(%i*theta_2)    //Total connected load(kVA)
pf_r = cos(phasemag(load)*%pi/180)                          //Resultant power
```

```
    factor
I_max = abs(load)*1000/(3**0.5*V*diver_f)                //Maximum distributor current per phase(A)
L_1 = L/2
V_per = 0.06*V/3**0.5                                    //Permissible voltage drop(V)

R_f = 0.734*10**-3                                       //Resistance(ohm/m)
X_f = 0.336*10**-3                                       //Reactance(ohm/m)
I_2f = P_2*10**3/(3**0.5*V)
I_1f = P_1*10**3/(3**0.5*V)
V_f = I_1f*L_1*(R_f*pf_1+X_f*sin(theta_1))+I_2f*L*(R_f*pf_2+X_f*sin(theta_2))
d_f = 9.0                                                //Overall conductor diameter(mm)
area_f = %pi*d_f**2/4                                    //Area of ferret conductor(mm^2)

R_R = 0.587*10**-3                                       //Resistance(ohm/m)
X_R = 0.333*10**-3                                       //Reactance(ohm/m)
I_2R = P_2*10**3/(3**0.5*V)
I_1R = P_1*10**3/(3**0.5*V)
V_R = I_1R*L_1*(R_R*pf_1+X_R*sin(theta_1))+I_2R*L*(R_R*pf_2+X_R*sin(theta_2))
d_R = 10.0
```

```
//Overall conductor diameter(mm)
area_R = %pi*d_R**2/4                                   //Area of rabbit conductor(mm^2)


// Result Section
if(V_f > V_per) then
    printf('Overall cross-sectional area of the 7/3.35 mm Rabbit ACSR conductors having overall conductor diameter of 10.0 mm = %.2f mm^2' ,area_R)
else
    printf('Overall cross-sectional area of the 7/3.00 mm Ferret ACSR conductors having overall conductor diameter of 9.0 mm = %.2f mm^2' ,area_f)
end

printf("END")
//End error, but doesn't affect functioning of code
```

Scilab code Exa 7.2 Example

```
// Variable Declaration

V = 400.0          //Voltage supplied(V)
i = 0.5            //Current per meter(A)
demand_f = 1.0     //Demand factor
diver_f = 1.0      //Diversity factor
L = 275.0          //Length of line(m)
pf = 0.9           //Power factor lagging

// Calculation Section
I = i*L
```

```
    //Current in distributor/phase(A)
theta = acos(pf)    //Power factor angle
V_per = 0.06*V/3**0.5    //Permissible voltage drop(V)

r_w = 0.985    //Resistance(ohm/km)
x_w = 0.341    //Reactance(ohm/km)
V_w = 0.5*i*(r_w*pf+x_w*sin(theta))*L**2*10**-3    //Voltage drop for Weasel(V)
d_w = 7.77    //Diameter of weasel conductor(mm)
area_w = %pi*d_w**2/4    //Area of weasel conductor(mm^2)

r_f = 0.734    //Resistance(ohm/km)
x_f = 0.336    //Reactance(ohm/km)
V_f = 0.5*i*(r_f*pf+x_f*sin(theta))*L**2*10**-3    //Voltage drop for Ferret(V)
d_f = 9.00    //Diameter of Ferret conductor(mm)
area_f = %pi*d_f**2/4    //Area of Ferret conductor(mm^2)

r_r = 0.587    //
```

```
Resistance(ohm/km)
x_r = 0.333                                              //Reactance(ohm/km)
V_r = 0.5*i*(r_r*pf+x_r*sin(theta))*L**2*10**-3   //Voltage drop for Rabbit(V)
d_r = 10.0                                               //Diameter of Rabbit conductor(mm)
area_r = %pi*d_r**2/4                                    //Area of Rabbit conductor(mm^2)

// Result Section
if(V_w < V_per) then
    printf('Overall cross-sectional area of the 7/2.59 mm Weasel ACSR conductors having overall conductor diameter of 7.77 mm = %.2f mm^2',area_w)
else if(V_f < V_per) then
    printf('Overall cross-sectional area of the 7/3.00 mm Ferret ACSR conductors having overall conductor diameter of 9.0 mm = %.2f mm^2',area_f)
else
    printf('Overall cross-sectional area of the 7/3.35 mm Rabbit ACSR conductors having overall conductor diameter of 10.0 mm = %.2f mm^2',area_r)
end
end
```

Scilab code Exa 7.3 Example

```

// Variable Declaration
```

```
V = 400.0          //Voltage supplied(V)
f = 50.0           //Frequency(Hz)
L = 300.0          //Length of line(m)
I_1 = 50.0         //Current at 100 m from feeding point(A)
pf_1 = 0.8         //Power factor at 100 m from feeding point
L_1 = 100.0        //Length of line upto feeding point(m)
I_2 = 25.0         //Current at 100 m from feeding point(A)
pf_2 = 0.78        //Power factor at 100 m from feeding point
L_2 = 200.0        //Length of line from feeding point to far end(m)
i = 0.2            //Distributed load current(A/metre)
v_drop = 15.0      //Permissible voltage drop

// Calculation Section
theta_1 = acos(pf_1)                  //Power factor angle for 50 A(radians)
theta_2 = acos(pf_2)                  //Power factor angle for 25 A(radians)

r_f = 0.734*10**-3                         //Resistance(ohm/m)
x_f = 0.336*10**-3                         //Reactance(ohm/m)
V_con_f = I_1*L_1*(r_f*pf_1+x_f*sin(theta_1))+I_2*L*(r_f*pf_2+x_f*sin(theta_2)) //Voltage drop at B due to concentrated loading(V)
V_dis_f = 0.5*i*r_f*(L_1+L_2)**2           //Voltage drop at B due to distributed loading(V)
V_f = V_con_f+V_dis_f                      //Total voltage drop(V)

r_r = 0.587*10**-3                         //Resistance(ohm/m)
```

```
x_r = 0.333*10**-3                         //Reactance(ohm/m)
V_con_r = I_1*L_1*(r_r*pf_1+x_r*sin(theta_1))+I_2*L*(r_r*pf_2+x_r*sin(theta_2)) //Voltage drop at B due to concentrated loading(V)
V_dis_r = 0.5*i*r_r*(L_1+L_2)**2           //Voltage drop at B due to distributed loading(V)
V_r = V_con_r+V_dis_r                      //Total voltage drop(V)

// Result Section
if(V_f < v_drop) then
    printf('Ferret ACSR conductors of size 7/3.00 mm having an overall conductor diameter of 9.0 mm is to be used')
    printf('Total voltage drop = %.2f V, which is within limit',V_f)
else
    printf('Rabbit ACSR conductors of size 7/3.35 mm having an overall conductor diameter of 10.0 mm is to be used')
    printf('Total voltage drop = %.2f V, which is within limit',V_r)
end
printf('\nNOTE : ERROR : In distributed load : current is 0.2 A/meter and not 0.25 A/meter as given in problem statement')
```

Scilab code Exa 7.4 Example

```
// Variable Declaration
P = 5.0          //Power of substation(MVA)
V_hv = 33.0      //High voltage(kV)
V_lv = 11.0      //Low voltage(kV)
f = 50.0         //Frequency(Hz)
```

```
P_1 = 0.5          //Minimum load(MW)
pf_1 = 0.85        //Lagging power factor of minimum load
P_2 = 2.8          //Maximum load(MW)
pf_2 = 0.78        //Lagging power factor of maximum load
pf_i = 0.9         //Lagging power factor of incoming current

// Calculation Section
theta_1 = acos(pf_1)
theta_2 = acos(pf_2)
theta_i = acos(pf_i)

load_react = P_1*tan(theta_1)*1000                       //Load reactive power(kVAR)
line_react = P_1*tan(theta_i)*1000                       //Reactive power supplied by line(kVAR)
rating_fix = load_react - line_react                        //kVAR rating of fixed capacitor bank(kVAR)

bank_react = P_2*(tan(theta_2)-tan(theta_i))*1000   //Reactive power to be supplied by capacitor banks(kVAR)
rating_swi = bank_react - rating_fix                        //Reactive power rating of switched unit(kVAR)

C_fix = rating_fix*10**-3/(3**0.5*V_lv**2*2*%pi*f)         //Capacitance for fixed bank
C_swi = rating_swi*10**-3/(3**0.5*V_lv**2*2*%pi*f)         //Capacitance for switched bank

// Result Section
printf('kVAR rating of fixed capacitors = %.1f kVAR',rating_fix)
```

```
printf('kVAR rating of switched capacitors = %.1f kVAR',rating_swi)
printf('Capacitance of fixed bank , C = %.2e F/phase',C_fix)
printf('Capacitance of switched bank , C = %.2e F/phase',C_swi)
```

Scilab code Exa 7.5 Example

```
// Variable Declaration
V = 400.0         //Voltage of induction motor(V)
f = 50.0          //Frequency(Hz)
I = 40.0          //Line current(A)
pf_1 = 0.78       //Lagging power factor of motor
pf_2 = 0.95       //Raised lagging power factor

// Calculation Section
theta_1 = acos(pf_1)                                      //Motor power factor angle(radians)
P_act_m = 3**0.5*V*I*pf_1*10**-3                          //Active power demand of motor(kW)
P_rea_m = P_act_m*tan(theta_1)                            //Reactive power demand of motor(kVAR)
theta_2 = acos(pf_2)                                      //Improved power factor angle(radians)
P_act_l = 3**0.5*V*I*pf_1*10**-3                          //Active power supplied by line(kW)
P_rea_l = P_act_m*tan(theta_2)                            //Reactive power supplied by line to motor(kVAR)
```

```
rating = P_rea_m - P_rea_l                                    //kVAR rating of capacitor bank(kVAR per phase)
I_C = rating*1000/(3**0.5*V)                                  //Current drawn by capacitor bank(A)
I_L = I*exp(%i*-theta_1)+I_C*exp(%i*90*%pi/180)      //Line current(A)
I_phase = I_C/3**0.5                                          //Phase current of delta connected capacitor bank(A)
C = I_phase/(V*2*%pi*f)                                       //Per phase capacitance of bank(micro-F/phase)


// Result Section
printf('kVAR rating of the bank = %.2f kVAR per phase',rating)
printf('Line current = %.2f  % .2f   A',abs(I_L),phasemag(I_L))
printf('Per phase capacitance of the bank , C = %.2e F/phase',C)
```

Scilab code Exa 7.6 Example

```
// Variable Declaration
P_1 = 250.0      //Load at unity power factor(kW)
pf_1 = 1         //Power factor
P_2 = 1500.0     //Load at 0.9 power factor(kW)
pf_2 = 0.9       //Lagging power factor
P_3 = 1000.0     //Load at 0.8 power factor(kW)
pf_3 = 0.8       //Lagging power factor
P_4 = 700.0      //Load at 0.78 power factor(kW)
```

```
pf_4 = 0.76       //Lagging power factor

// Calculation Section
theta_1 = acos(pf_1)
theta_2 = acos(pf_2)
theta_3 = acos(pf_3)
theta_4 = acos(pf_4)
kW_T = P_1+P_2+P_3+P_4                      //Total kW carried by feeder(kW)
kVAR_T = P_1*tan(theta_1)+P_2*tan(theta_2)+P_3*tan(theta_3)+P_4*tan(theta_4)
pf_feed = cos(atan(kVAR_T/kW_T))
feeder_KVA = (kW_T**2+kVAR_T**2)**0.5     //Feeder kVA
feeder_kW = feeder_KVA                      //Load at unity pf(kW)


// Result Section
printf('Feeder power factor = %.3f lagging' ,pf_feed)
printf('Load at unity power factor = %.f kW' ,feeder_kW)
printf('\nNOTE : ERROR : The load data should be 700 kW at 0.76 pf lagging instead of 700 kW at 0.78 lagging')
```

Scilab code Exa 7.8 Example

```
// Variable Declaration
V = 400.0         //Voltage(V)
f = 50.0          //Frequency(Hz)
HP_1 = 75.0       //Power(H.P)
HP_2 = 25.0       //Power(H.P)
HP_3 = 10.0       //Power(H.P)
```

```
pf_1 = 0.75      //Power factor at 3/4 load
pf_2 = 0.78      //Power factor at 4/5 load
pf_3 = 0.8       //Power factor at full load
pf_4 = 0.9       //Lagging power factor improved
pf_5 = 0.74      //Power factor of 2nd motor at 2/3 of full load
pf_6 = 0.8       //Power factor of 3rd motor at full load

// Calculation Section
theta_1 = acos(pf_1)
theta_2 = acos(pf_2)
theta_3 = acos(pf_3)
S_1P = (0.75*HP_1*746*10**-3/pf_1)*exp(%i*theta_1)      //kVA demanded by first motor(kVA)
S_2P = (0.8*HP_2*746*10**-3/pf_2)*exp(%i*theta_2)       //kVA demanded by second motor(kVA)
S_3P = (HP_3*746*10**-3/pf_3)*exp(%i*theta_3)           //kVA demanded by third motor(kVA)
S_TP = S_1P + S_2P + S_3P                               //Total kVA demanded by all loads(kVA)
pf_l_wc = cos(phasemag(S_TP)*%pi/180)                   //Line power factor without capacitive correction
kW_T = real(S_TP)                                       //Total kW demanded by load(kW)
kVAR_T = imag(S_TP)                                     //Total lagging kVAR demanded by loads(kVAR)
theta_4 = acos(pf_4)
P_react = kW_T*tan(theta_4)                             //Reactive power supplied by line for 0.9 pf(kVAR)
power = kVAR_T - P_react                                //Reactive power supplied by capacitor bank(kVAR)
```

```

theta_5 = acos(pf_5)
theta_6 = acos(pf_6)
S_2L = (2*HP_2*746*10**-3/(3*pf_5))*exp(%i*theta_5)
     //kVA demanded by second motor(kVA)
S_3L = (HP_3*746*10**-3/pf_3)*exp(%i*theta_3)
           //kVA demanded by third motor(kVA)
S_TL = S_2L + S_3L
                                                 //Total
   kVA demanded during lean period(kVA)
S_line = real(S_TL) - complex(0,power-imag(S_TL))
             //kVA supplied by line(kVA)
pf_line = cos(phasemag(S_line)*%pi/180)
                      //Line power factor

// Result Section
printf('Line power factor with capacitor bank
   connected during lean period = %.2f leading' ,
   pf_line)
```

Chapter 8

ELEMENTS OF ELECTRIC POWER GENERATION

Scilab code Exa 8.1 Example

```
// Variable Declaration
w = 0.8      //Coal to be burnt for every kWh of
    electric energy(kg)
C = 5000     //Calorific value of coal(kilo-calories/
    kg)

// Calculation Section
heat_energy = C*w/860         //Heat energy of
    combustion of given coal(kWh)
efficiency = 1/heat_energy   //Overall efficiency


// Result Section
printf('Overall efficiency of the plant = %.3f' ,
    efficiency)
```

Scilab code Exa 8.2 Example

```
// Variable Declaration
P = 250.0          //Power(MW)
C = 6100.0         //Calorific value(kcal/kg)
n_1 = 0.9          //Plant runs at full load
h_1 = 20.0         //Time for full load(hour)
n_2 = 0.75         //Plant runs at full load
h_2 = 4.0          //Time for full load(hour)
n_t = 0.3          //Thermal efficiency
n_g = 0.93         //Generator efficiency

// Calculation Section
E_T = (P*n_1*h_1+P*n_2*h_2)*1000     //Total electric
    energy produced by plant in a day(kWh)
efficiency = n_t * n_g               //Overall
   efficiency of the plant
heat_energy = E_T*860/efficiency     //Heat energy of
    combustion of coal(kcal)
coal_requ = heat_energy/C            //Daily coal
   requirement(kg)
coal_requ_ton = coal_requ*10**-3     //Daily coal
   requirement(tonnes)

// Result Section
printf('Daily coal requirement = %.2e kg = %.f
   tonnes' ,coal_requ,coal_requ_ton)
```

Scilab code Exa 8.3 Example

```
// Variable Declaration
Q = 1.0            //Water discharge(m^3/sec)
h = 200.0          //Height(m)
n_h = 0.85         //Hydraulic efficiency
n_e = 0.95         //Electric efficiency
```

```

// Calculation Section
n = n_h*n_e                 //Overall efficiency
P = (736.0/75)*Q*h*n        //Electrical power available
    (kW)
E = P*1.0                   //Energy available in an
    hour(kWh)

// Result Section
printf('Electrical power available = %.2f kW',P)
printf('Energy available in an hour = %.2f kWh',E)
```

Scilab code Exa 8.4 Example

```
// Variable Declaration
Ad = 6.0*10**6          //Reservoir capacity(m^3)
h = 150.0               //Head(m)
n = 0.78                //Overall efficiency
P = 25.0*10**6          //Power(Watt)
t = 4.0                 //Supply time(hour)

// Calculation Section
AX = P*75*3600*t/(736*h*n*1000)    //unit(m^3)
X_d = AX/Ad*100                    //Fall in
    reservoir level(%)

// Result Section
printf('Percentage fall in reservoir level = %.2f
    percent',X_d)
```

Scilab code Exa 8.5 Example

```
// Variable Declaration
```

```
X_s = 1.0        //Synchronous reactance of generator(p.u)
V_b = 1.0        //Terminal voltage of generator=voltage of infinite bus(p.u)
P_G = 0.5        //Real power output at unity pf(p.u)


// Calculation Section
I = P_G/V_b                            //Generator current(p.u)
E = complex(V_b,I*X_s)                 //Excitation emf of finite machine(p.u)
delta = phasemag(E)                    //Power angle = angle b/w E & V_b(degree)

P_Gn = P_G/2                           //Real power o/p when steam i/p is halved(p.u)
sin_delta_n = P_Gn*X_s/(abs(E)*V_b)
delta_n = asin(sin_delta_n)     //New power angle(radian)
E_n = abs(E)*exp(%i*delta_n)  //Excitation emf of finite machine with new angle(p.u)
I_n = (E_n-V_b)/complex(0,X_s)         //Current when steam i/p is halved(p.u)
pf_n = cos(phasemag(I_n)*%pi/180)     //Power factor when steam i/p is halved

P_po = abs(E)*V_b/X_s                  //Pull out power(p.u)

stiff_a = abs(E)*V_b/X_s*cos(phasemag(E)*%pi/180)     //Electrical stiffness in case(a) (p.u/radian)
stiff_b = abs(E)*V_b/X_s*cos(phasemag(I_n)*%pi/180)    //Electrical stiffness in case(b) (p.u/radian)

// Result Section
printf('Case(a) :')
```

```
printf('Excitation voltage of finite machine , E = %.2f %.2f p.u',abs(E),delta)
printf('Power angle = %.2f ',delta)
printf('\nCase(b) :')
printf('Current if steam input is reduced to half , I_n = %.3f %.2f p.u',abs(I_n),phasemag(I_n))
printf('Power factor if steam input is reduced to half = %.2f lagging',pf_n)
printf('Power angle if steam input is reduced to half = %.2f ',delta_n*180/%pi)
printf('\nCase(c) :')
printf('Pull out power = %.2f p.u',P_po)
printf('\nCase(d) :')
printf('Electrical stiffness for case(a) = %.1f p.u/radian',stiff_a)
printf('Electrical stiffness for case(b) = %.3f p.u/radian',stiff_b)
```

Scilab code Exa 8.6 Example

```
// Variable Declaration
X_s = 1.1        //Synchronous reactance of generator(p.u)
V_b = 1.0        //Terminal voltage of generator=voltage of infinite bus(p.u)
E = 1.25         //Excitation emf of finite machine(p.u)
P_G = 0.3        //Active power output(p.u)
dec = 0.25       //Excitation is decreased

// Calculation Section
sin_delta = P_G*X_s/(E*V_b)
delta = asin(sin_delta)                 //Power angle(radian)
Q_G = V_b/X_s*(E*cos(delta)-V_b)        //Reactive
```

```
    power output(p.u)

E_n = (1-dec)*E                              //New excitation emf of finite machine(p.u)
P_Gn = P_G                                   //New active power output(p.u)
sin_delta_n = P_G*X_s/(E_n*V_b)
delta_n = asin(sin_delta_n)                  //New power angle(radian)
Q_Gn = V_b/X_s*(E_n*cos(delta_n)-V_b)   //New reactive power output(p.u)


// Result Section
printf('Case(a) :')
printf('Power angle = %.2f  ',delta*180/%pi)
printf('Reactive power output , Q_G = %.3f p.u',Q_G)
printf('\nCase(b) :')
printf('Active power if excitation is decreased , P_Gn = %.1f p.u',P_Gn)
printf('Reactive power if excitation is decreased , Q_Gn = %.3f p.u',Q_Gn)
printf('Power angle if excitation is decreased = %.2f  ',delta_n*180/%pi)
```

Scilab code Exa 8.7 Example

```
// Variable Declaration
X_s = 1.05       //Synchronous reactance of generator(p.u)
V_b = 0.95       //Terminal voltage of generator=voltage of infinite bus(p.u)
X_L = 0.1        //Reactance of link(p.u)
E = 1.2          //Excitation emf of finite machine(p
```

```
.u)
P_G = 0.15        //Active power output(p.u)
inc = 1           //Turbine torque increased

// Calculation Section
sin_delta = P_G*(X_s+X_L)/(E*V_b)
delta = asin(sin_delta)                       //Power angle(radian)
Q_G = V_b/(X_s+X_L)*(E*cos(delta)-V_b)        //Reactive power output(p.u)

P_Gn = (1+inc)*P_G                            //New active power output(p.u)
sin_delta_n = P_Gn*(X_s+X_L)/(E*V_b)
delta_n = asin(sin_delta_n)                   //Power angle(radian)
Q_Gn = V_b/(X_s+X_L)*(E*cos(delta_n)-V_b)     //Reactive power output(p.u)
P_change = (P_Gn-P_G)/P_G*100                 //Change in active power output(%)
Q_change = (Q_Gn-Q_G)/Q_G*100                 //Change in reactive power output(%)

// Result Section
printf('Change in active power supplied by generator = %.f percent',P_change)
printf('Change in reactive power supplied by generator = %.2f percent',Q_change)
```

Scilab code Exa 8.8 Example

```
// Variable Declaration
X_s = 6.0         //Synchronous reactance of alternator(ohms/phase)
pf = 0.8          //Lagging power factor
```

```
P_G = 5.0          //Power delivered(MW)
V = 11.0           //Voltage of infinite bus(kV)

// Calculation Section
delta = acos(pf)
I = P_G*1000/(3**0.5*V*pf)*(pf - complex(0,sin(delta)))       //Alternator current(A)
V_b = V*10**3/3**0.5                                              //Voltage of infinite bus(V/phase)
E = complex(7531.79669352,1574.59164324)                     //Initial excitation voltage(V)
pf_n = 1.0                                                        //New power factor
P_Gn = P_G                                                        //New power delivered(MW)
I_n = P_Gn*1000/(3**0.5*V*pf_n)                                 //Alternator current(A)
E_n = complex(V_b,I_n*X_s)                                        //New excitation voltage(V)
excitation_change = (abs(E)-abs(E_n))/abs(E)*100              //Percentage change in excitation(%)

// Result Section
printf('Percentage change in excitation = %.2f percent',excitation_change)
```

Chapter 9

LOAD FLOW STUDIES

Scilab code Exa 9.1 Example

```
// Variable Declaration
Y_s12 = complex(2.96,-20.16)      //Line admittance b
    /w buses 1 & 2(*10^-3 mho)
Y_p12 = complex(0,0.152)          //Line admittance b
    /w buses 1 & 2(*10^-3 mho)
Y_s15 = complex(2.72,-18.32)      //Line admittance b
    /w buses 1 & 5(*10^-3 mho)
Y_p15 = complex(0,0.185)          //Line admittance b
    /w buses 1 & 5(*10^-3 mho)
Y_s23 = complex(3.0,-22.8)        //Line admittance b
    /w buses 2 & 3(*10^-3 mho)
Y_p23 = complex(0,0.110)          //Line admittance b
    /w buses 2 & 3(*10^-3 mho)
Y_s25 = complex(1.48,-10.30)      //Line admittance b
    /w buses 2 & 5(*10^-3 mho)
Y_p25 = complex(0,0.312)          //Line admittance b
    /w buses 2 & 5(*10^-3 mho)
Y_s34 = complex(2.96,-20.16)      //Line admittance b
    /w buses 3 & 4(*10^-3 mho)
Y_p34 = complex(0,0.152)          //Line admittance b
    /w buses 3 & 4(*10^-3 mho)
```

```
Y_s45 = complex(3.0,-22.8)          //Line admittance b/w buses 4 & 5(*10^-3 mho)
Y_p45 = complex(0,0.110)            //Line admittance b/w buses 4 & 5(*10^-3 mho)


// Calculation Section
Y_s13 = complex(0,0)                //Line admittance b/w buses 1 & 3(*10^-3 mho)
Y_p13 = complex(0,0)                //Line admittance b/w buses 1 & 3(*10^-3 mho)
Y_s14 = complex(0,0)                //Line admittance b/w buses 1 & 4(*10^-3 mho)
Y_p14 = complex(0,0)                //Line admittance b/w buses 1 & 4(*10^-3 mho)
Y_11 = (Y_s12+Y_s13+Y_s14+Y_s15)+(Y_p12+Y_p13+Y_p14+Y_p15)
Y_12 = -Y_s12
Y_13 = -Y_s13
Y_14 = -Y_s14
Y_15 = -Y_s15

Y_s21 = Y_s12
Y_p21 = Y_p12
Y_s24 = complex(0,0)                //Line admittance b/w buses 2 & 4(*10^-3 mho)
Y_p24 = complex(0,0)                //Line admittance b/w buses 2 & 4(*10^-3 mho)
Y_21 = Y_12
Y_22 = (Y_s21+Y_s23+Y_s24+Y_s25)+(Y_p21+Y_p23+Y_p24+Y_p25)
Y_23 = -Y_s23
Y_24 = -Y_s24
Y_25 = -Y_s25

Y_s31 = Y_s13
Y_p31 = Y_p13
Y_s32 = Y_s23
```

```
Y_p32 = Y_p23
Y_s35 = complex(0,0)              //Line admittance b/w buses 2 & 4(*10^-3 mho)
Y_p35 = complex(0,0)              //Line admittance b/w buses 2 & 4(*10^-3 mho)
Y_33 = (Y_s31+Y_s32+Y_s34+Y_s35)+(Y_p31+Y_p32+Y_p34+Y_p35)
Y_34 = -Y_s34
Y_35 = -Y_s35
Y_31 = Y_13
Y_32 = Y_23
Y_33 = (Y_s31+Y_s32+Y_s34+Y_s35)+(Y_p31+Y_p32+Y_p34+Y_p35)
Y_34 = -Y_s34
Y_35 = -Y_s35

Y_s41 = Y_s14
Y_p41 = Y_p14
Y_s42 = Y_s24
Y_p42 = Y_p24
Y_s43 = Y_s34
Y_p43 = Y_p34
Y_41 = Y_14
Y_42 = Y_24
Y_43 = Y_34
Y_44 = (Y_s41+Y_s42+Y_s43+Y_s45)+(Y_p41+Y_p42+Y_p43+Y_p45)
Y_45 = -Y_s45

Y_s51 = Y_s15
Y_p51 = Y_p15
Y_s52 = Y_s25
Y_p52 = Y_p25
Y_s53 = Y_s35
Y_p53 = Y_p35
Y_s54 = Y_s45
Y_p54 = Y_p45
Y_51 = Y_15
```

```
Y_52 = Y_25
Y_53 = Y_35
Y_54 = Y_45
Y_55 = (Y_s51+Y_s52+Y_s53+Y_s54)+(Y_p51+Y_p52+Y_p53+Y_p54)

Y_bus = [[Y_11, Y_12, Y_13, Y_14, Y_15],
         [Y_21, Y_22, Y_23, Y_24, Y_25],
         [Y_31, Y_32, Y_33, Y_34, Y_35],
         [Y_41, Y_42, Y_43, Y_44, Y_45],
         [Y_51, Y_52, Y_53, Y_54, Y_55]]

// Result Section
printf('The Y bus matrix for the five-bus system is :\n')
disp(Y_bus)
```

Scilab code Exa 9.2 Example

```
// Variable Declaration
V_1 = complex(1.04,0)            //Voltage at bus 1(p.u)
S_D1 = complex(0.55,0.15)        //Power at bus 1(p.u)
S_D2 = complex(1.0,0.3)          //Power at bus 2(p.u)
Y_11 = complex(0.988,-9.734)     //Admittance at bus 1(p.u)
Y_22 = Y_11                      //Admittance at bus 2(p.u)
Y_12 = complex(-0.988,9.9)       //Admittance b/w bus 1 & 2(p.u)
Y_21 = Y_12                      //Admittance b/w bus 2 & 1(p.u)

```

```
// Calculation Section
V_2_0 = complex(1,0)                                  //Initial value of V_2
S_2 = complex(-1,0.3)                                 //P_2+j*Q_2
V_2_1 =  (1/Y_22)*(S_2/conj(V_2_0)-Y_21*V_1)
V_2_2 =  (1/Y_22)*(S_2/conj(V_2_1)-Y_21*V_1)
V_2_3 =  (1/Y_22)*(S_2/conj(V_2_2)-Y_21*V_1)
V_2_4 =  (1/Y_22)*(S_2/conj(V_2_3)-Y_21*V_1)
V_2_5 =  (1/Y_22)*(S_2/conj(V_2_4)-Y_21*V_1)
V_2 = V_2_5                                           //Voltage 2(p.u)
S_1_con = conj(V_1)*Y_11*V_1 + conj(V_1)*Y_12*V_2     //Conjugate of slack bus net power
S_1 = conj(S_1_con)
S_G1 = S_1 + S_D1                                     //Generated power at bus 1(p.u)
P_L = real(S_G1) - (real(S_D1) + real(S_D2))          //Real power loss(p.u)
Q_L = imag(S_G1) - (imag(S_D1) + imag(S_D2))          //Reactive power loss(p.u)

// Result Section
printf('Voltage at bus 2 , V_2 = %.4f  %.2f  p.u',abs(V_2),phasemag(V_2))
printf('Generated power at bus 1 , S_G1 = (%.2f + j%.3f) p.u' ,real(S_G1),imag(S_G1))
printf('Real power loss in the system = %.2f p.u' ,P_L)
printf('Reactive power loss in the system = %.3f p.u' ,Q_L)
```

Chapter 10

POWER SYSTEM ECONOMICS

Scilab code Exa 10.1 Example

```

// Variable Declaration
max_dm_kW = 150.0        //Maximum demand(kW)
pf = 0.85                //Average power factor
rate = 90.0              //Cost of maximum demand(Rs/
    kVA)
E_rate = 0.3             //Cost of energy consumed(Rs
    )
lf = 0.65                //Annual load factor

// Calculation Section
max_dm_kVA = max_dm_kW/pf                  //Maximum
    demand(kVA)
annual_chg_kVA = rate*max_dm_kVA           //Annual
    fixed charges based on max demand(Rs)
E_kWh = lf*365*24*max_dm_kW                //Energy
    consumed per annum(kWh)
annual_E_chg = E_kWh*E_rate                //Annual
    energy charges(Rs)
```

```
annual_elect_charge = annual_chg_kVA + annual_E_chg
        //Annual electricity charge to be paid(Rs)

// Result Section
printf('Annual electricity charges to be paid by
    consumer = Rs %.2f',annual_elect_charge)
```

Scilab code Exa 10.2 Example

```

// Variable Declaration
P = 75.0                    //Power(kW)
cost_plant = 3000.0         //Cost of plant(Rs/kW)
cost_td = 30.0*10**5        //Cost of transmission &
    distribution(Rs)
interest = 0.15             //Interest,insurance charges
    (/annum)
depreciation = 0.05         //Depreciation(/annum)
cost_fix_mt = 4.0*10**5     //Fixed maintainance(Rs)
cost_var_mt = 6.0*10**5     //Variable maintainance(Rs)
cost_fuel = 10.0*10**6      //Fuel cost(Rs/annum)
cost_opr = 3.0*10**6        //Operation cost(Rs/annum)
max_demand = 70.0           //Maximum demand(MW)
df = 1.6                    //Diversity factor b/w
    consumers
lf = 0.6                    //Annual load factor
dividend = 10**6            //Dividend to shareholders(
    Rs/annum)
per_L = 0.10                //Total energy loss(% of
    generated energy)


// Calculation Section
cost = cost_plant*P*1000
                                                //Cost of
```

```
plant(Rs)
per_value = interest+depreciation                                   //Total interest & depreciation(/annum)
cost_fix_ann = (cost+cost_opr)*per_value+cost_fix_mt+dividend    //Total fixed cost(Rs)
cost_var_ann = cost_fuel+cost_opr+cost_var_mt                       //Total running cost(Rs)
E_gen_ann = max_demand*1000*24*365*lf                               //Energy generated per annum(kWh)
E_loss = per_L*E_gen_ann                                            //Energy losses(kWh)
E_sold = E_gen_ann - E_loss                                         //Energy sold (kWh)
sum_max_demand = df*max_demand*1000                                 //Sum of maximum demand of consumers(kW)
charge_max_demand = cost_fix_ann/sum_max_demand                     //Charge to consumers per kW of max demand per year(Rs)
charge_energy = cost_var_ann/E_sold*100                             //Charge for energy(paise per kWh)


// Result Section
printf('Two-part tariff is :')
printf('Rs %.2f per kW of maximum demand per year + %.1f paise per kWh consumed',charge_max_demand,charge_energy)
```

Scilab code Exa 10.3 Example

```

// Variable Declaration
P_D = 500.0      //Total load(MW)
b_1 = 15.0       //Beta value of controllable thermal plant C1
g_1 = 0.012      //Gamma value of controllable thermal plant C1
b_2 = 16.0       //Beta value of controllable thermal plant C2
g_2 = 0.018      //Gamma value of controllable thermal plant C2
b_3 = 19.0       //Beta value of controllable thermal plant C3
g_3 = 0.020      //Gamma value of controllable thermal plant C3


// Calculation Section
l = (P_D+((b_1/(2*g_1))+(b_2/(2*g_2))+(b_3/(2*g_3))))/((1/(2*g_1))+(1/(2*g_2))+(1/(2*g_3))) //Lambda value which is a Lagrange multiplier
P_G1 = (l - b_1)/(2*g_1)                          //(MW)
P_G2 = (l - b_2)/(2*g_2)                          //(MW)
P_G3 = (l - b_3)/(2*g_3)                          //(MW)
C1 = 1500.0 + b_1*P_G1 + g_1*P_G1**2    //Fuel cost of plant C1(Rs/hr)
C2 = 2000.0 + b_2*P_G2 + g_2*P_G2**2    //Fuel cost of plant C2(Rs/hr)
C3 = 1000.0 + b_3*P_G3 + g_3*P_G3**2    //Fuel cost of plant C3(Rs/hr)
C = C1 + C2 + C3                        //Total fuel cost(Rs/hr)


// Result Section
printf('Value of  from equation(10.14) = %.3f',l)
printf('Optimal scheduling of thermal plant C1 = %.2f MW',P_G1)
```

```
printf('Optimal scheduling of thermal plant C2 = %.2
   f MW' ,P_G2)
printf('Optimal scheduling of thermal plant C3 = %.2
   f MW' ,P_G3)
printf('Total cost , C = Rs %.2f/hr' ,C)
```

Chapter 12

OVER VOLTAGE TRANSIENTS IN POWER SYSTEMS AND PROTECTION

Scilab code Exa 12.1 Example

```

// Variable Declaration
V_i = 100.0          //Incident voltage(kV)
Z_1 = 400.0          //Surge impedance(ohm)
Z_2 = 350.0          //Surge impedance(ohm)


// Calculation Section
beta = 2*Z_2/(Z_1+Z_2)       //Refraction
    coeffeicient of voltage
alpha = (Z_2-Z_1)/(Z_1+Z_2) //Reflection
    coeffeicient of voltage
V_t = beta*V_i               //Refracted voltage(kV)
V_r = alpha*V_i              //Reflected voltage(kV)
I_t = V_t/Z_2*1000           //Refracted current(A)
```

```
I_r = -(V_r/Z_1)*1000        //Reflected current(A)


// Result Section
printf('Reflected voltage , V_r = %.1f kV' ,V_r)
printf('Refracted voltage , V_t = %.1f kV' ,V_t)
printf('Reflected current , I_r = %.1f A' ,I_r)
printf('Refracted current , I_t = %.1f A' ,I_t)
```

Scilab code Exa 12.2 Example

```

// Variable Declaration
V_i = 100.0            //Incident voltage(kV)
Z_1 = 400.0            //Surge impedance(ohm)
Z_21 = 350.0           //Surge impedance of line connected at T(ohm)
Z_22 = 50.0            //Surge impedance of cable connected at T(ohm)


// Calculation Section
Z_2 = Z_21*Z_22/(Z_21+Z_22)       //Surge impedance(ohm)
V_t = 2*Z_2*V_i/(Z_1+Z_2)         //Refracted voltage(kV)
V_r = (Z_2-Z_1)*V_i/(Z_1+Z_2)     //Reflected voltage(kV)
I_t1 = V_t/Z_21*1000              //Refracted current in Z_21(A)
I_t2 = V_t/Z_22*1000              //Refracted current in Z_22(A)
I_r = -(V_r/Z_1)*1000             //Reflected current in Z_1(A)

```

```
// Result Section
printf('Refracted voltage , V_t = %.2f kV' ,V_t)
printf('Refracted current in overhead line , I_t1 = %.2f A',I_t1)
printf('Refracted current in underground cable , I_t2 = %.2f A' ,I_t2)
```

Scilab code Exa 12.3 Example

```


// Variable Declaration
V_i = 100.0          //Incident voltage(kV)
Z_1 = 400.0          //Surge impedance of overhead line(ohm)
Z_2 = 50.0           //Surge impedance of underground cable(ohm)


// Calculation Section
beta = 2*Z_2/(Z_1+Z_2)       //Refraction coeffeicient of voltage
alpha = (Z_2-Z_1)/(Z_1+Z_2)  //Reflection coeffeicient of voltage
V_t = beta*V_i               //Refracted voltage(kV)
V_r = alpha*V_i              //Reflected voltage(kV)
I_t = V_t/Z_2*1000           //Refracted current(A)
I_r = -(V_r/Z_1)*1000        //Reflected current(A)



// Result Section
printf('Reflected voltage , V_r = %.1f kV' ,V_r)
printf('Refracted voltage , V_t = %.1f kV' ,V_t)
```

```
printf('Reflected current , I_r = %.1f A' ,I_r)
printf('Refracted current , I_t = %.1f A' ,I_t)
```

Scilab code Exa 12.5 Example

```


// Variable Declaration
R = 74.0*10**-6            //Resistance of overhead
    line(ohm/meter)
L = 1.212*10**-6           //Inductance of overhead
    line(H/meter)
C = 9.577*10**-12          //Capacitance of overhead
    line(F/meter)


// Calculation Section
Z_0 = (L/C)**0.5           //Surge impedance of line(
    ohm)
a = R/(2*Z_0)
x_1 = log(2)/a       //Distance to be travelled(m)


// Result Section
printf('The distance the surge must travel to
    attenuate to half value = %.2e meter = %.2e km' ,
    x_1,x_1*10**-3)
```

Scilab code Exa 12.7 Example

```

// Variable Declaration
V_i = 2000.0          //Incident voltage(kV)
```

```
Z = 300.0              //Surge impedance(ohm)
V_p = 1200.0           //Arrester protection level(kV)

// Calculation Section
I_surge = V_i/Z            //Surge current(kA)
V_oc = 2*V_i               //Open-circuit voltage(kV)
I_A = (V_oc-V_p)/Z         //Current through the arrestor(kA)
I_r = I_A - I_surge        //Reflected current in line(kA)
V_r = -I_r*Z               //Reflected voltage of line(kV)
V_t = V_p                  //Refracted voltage into arrestor(kV)
V_r_coeff = V_r/V_i        //Reflected coefficient of voltage
V_t_coeff = V_t/V_i        //Refracted coefficient of voltage
R_a = V_p/I_A              //Arrestor resistance(ohm)


// Result Section
printf('Case(a) :')
printf('Current flowing in line before the surge voltage reaches the arrestor terminal = %.2f kA',I_surge)
printf('\nCase(b) :')
printf('Current through the arrestor , I_A = %.2f kA',I_A)
printf('\nCase(c) :')
printf('Refraction coefficient of voltage at arrestor terminals = %.1f ',V_t_coeff)
printf('Reflection coefficient of voltage at arrestor terminals = %.1f ',V_r_coeff)
printf('\nCase(d) :')
printf('Value of arrestor resistance = %.1f ohm',R_a)
```

Chapter 13

SHORT CIRCUIT PHENOMENA

Scilab code Exa 13.1 `Example`

```

// Variable Declaration
kv_gA = 11.0          //Voltage rating of generator A(kV)
MVA_gA = 40.0         //MVA rating of generator A
x_gA = 0.12           //Reactance of generator A(p.u)
kv_gB = 11.0          //Voltage rating of generator B(kV)
MVA_gB = 20.0         //MVA rating of generator B
x_gB = 0.08           //Reactance of generator B(p.u)
kv_Tlv = 11.0         //Low-voltage winding of transformer(kV)
kv_Thv = 66.0         //High-voltage winding of transformer(kV)
x_T = 0.10            //Reactance of Transformer(p.u)
kv_f = 66.0           //Feeder voltage(kV)
x_f = 30.0            //Reactance of feeder(ohm)


```

```
// Calculation Section
MVA_base = 75.0    //Base MVA
kv_base_lv = 11.0    //Base voltage on LT side(kV)
kv_base_hv = 66.0    //Base voltage on HT side(kV)
x_gA_new = x_gA*(MVA_base/MVA_gA)    //New Reactance of generator A(p.u)
x_gB_new = x_gB*(MVA_base/MVA_gB)    //New Reactance of generator B(p.u)
x_f_new = x_f*(MVA_base/kv_base_hv**2)    //New reactance of feeder(p.u)

x_eq = x_T+(x_gA_new*x_gB_new/(x_gA_new+x_gB_new))    //Equivalent reactance(p.u)
V_f = kv_Thv/kv_base_hv    //Fault voltage by applying Thevenin's Theorem at FF(p.u)
I_f = V_f/complex(0,x_eq)    //Fault current(A)
I_f_ht = I_f*(MVA_base*1000/(3**0.5*kv_base_hv))    //Fault current on HT side(A)
I_f_lt = I_f_ht*kv_base_hv/kv_base_lv    //Fault current on LT side(A)
MVA_fault = V_f*MVA_base/x_eq    //Fault MVA
I_A = I_f*x_gB_new/(x_gA_new+x_gB_new)    //Current in generator A(p.u)
I_A1 = I_A*MVA_base*1000/(3**0.5*kv_base_lv)    //Current in generator A(A)
I_B = I_f*x_gA_new/(x_gA_new+x_gB_new)    //Current in generator B(p.u)
I_B1 = I_B*MVA_base*1000/(3**0.5*kv_base_lv)    //Current in generator B(A)

x_eq2 = x_f_new+x_T+(x_gA_new*x_gB_new/(x_gA_new+x_gB_new))    //Equivalent reactance(p.u)
```

```
I_f2 = V_f/complex(0,x_eq2)                                  //Fault current(p.u)
I_f_ht2 = I_f2*(MVA_base*1000/(3**0.5*kv_base_hv))           //Fault current on HT side(A)
MVA_fault2 = V_f*MVA_base/x_eq2                              //Fault MVA
I_A_pu = I_f2*x_gB_new/(x_gA_new+x_gB_new)                   //Current in generator A(p.u)
I_A2 = I_A_pu*MVA_base*1000/(3**0.5*kv_base_lv)              //Current in generator A(A)
I_B_pu = I_f2*x_gA_new/(x_gA_new+x_gB_new)                   //Current in generator B(p.u)
I_B2 = I_B_pu*MVA_base*1000/(3**0.5*kv_base_lv)              //Current in generator B(A)


// Result Section
printf('Case(a) :')
printf('Fault MVA for symmetric fault at the high voltage terminals of transformer = %.2f MVA',MVA_fault)
printf('Fault current shared by generator A , I_A = %.2fj A',imag(I_A1))
printf('Fault current shared by generator B , I_B = %.2fj A',imag(I_B1))
printf('\nCase(b) :')
printf('Fault MVA for symmetric fault at the load end of the feeder = %.2f MVA',MVA_fault2)
printf('Fault current shared by generator A , I_A = %.2fj A',imag(I_A2))
printf('Fault current shared by generator B , I_B = %.2fj A',imag(I_B2))
```

Scilab code Exa 13.2 Example

```

// Variable Declaration
MVA_base = 100.0     //Base MVA
x1 = 0.15            //Reactance b/w F & B(p.u) . (
    Refer textbook diagram for marking)
x2 = 0.1             //Reactance b/w F & B(p.u)
x3 = 0.18            //Reactance b/w B & C(p.u)
x4 = 0.1             //Reactance b/w B & F(p.u)
x5 = 0.05            //Reactance b/w F & C(p.u)
x6 = 0.05            //Reactance b/w F & C(p.u)
x7 = 0.1             //Reactance b/w C & F(p.u)
x8 = 0.12            //Reactance b/w C & F(p.u)


// Calculation Section
V_f = 1.0            //Fault voltage by applying
    Thevenin's Theorem at FF(p.u)
x1_eq = x1+x2
x2_eq = x7+x8
x3_eq = x5*x6/(x5+x6)
x4_eq = x3*x4/(x3+x4+x3_eq)
x5_eq = x4*x3_eq/(x3+x4+x3_eq)
x6_eq = x3*x3_eq/(x3+x4+x3_eq)
x7_eq = (x1_eq+x4_eq)*(x2_eq+x6_eq)/(x1_eq+x4_eq+
    x2_eq+x6_eq)
X_eq = x7_eq+x5_eq               //Equivalent
    reactance
MVA_SC = V_f*MVA_base/X_eq       //Short circuit MVA
    at A


// Result Section
```

```
printf('Rating of the circuit breaker at the location A = %.1f MVA',MVA_SC)
printf('\nNOTE : ERROR : Delta to star reactance conversion mistake in textbook')
```

Scilab code Exa 13.3 Example

```


// Variable Declaration
x = 1.2                   //Reactance of interconnector(ohm per phase)
kv = 33.0                 //Voltage of bus-bars(kV)
SC_MVA1 = 3000.0          //Short-circuit MVA at bus-bar of first station(MVA)
SC_MVA2 = 2000.0          //Short-circuit MVA at bus-bar of second station(MVA)


// Calculation Section
MVA_base = 3000.0                    //Base MVA
kv_base = 33.0                       //Base kV
x_c = x*(MVA_base/kv_base**2)        //Cable reactance(p.u)
x1 = MVA_base/SC_MVA1                //Reactance b/w e.m.f source & bus-bars for station 1(p.u)
x2 = MVA_base/SC_MVA2                //Reactance b/w e.m.f source & bus-bars for station 2(p.u)
V_f = 1.0                            //Fault voltage by applying Thevenin's Theorem at FF(p.u)
X_eq1 = x1*(x_c+x2)/(x1+x_c+x2)      //Thevenin reactance for short-circuit at bus bars at station 1(p.u)
SC_MVA1_poss = V_f*MVA_base/X_eq1    //Possible short-circuit at station 1(MVA)
```

```
X_eq2 = x2*(x_c+x1)/(x1+x_c+x2)     //Thevenin reactance for short-circuit at bus bars at station 2(p.u)
SC_MVA2_poss = V_f*MVA_base/X_eq2   //Possible short-circuit at station 2(MVA)


// Result Section
printf('Possible short-circuit MVA at station 1 = %.2f MVA' ,SC_MVA1_poss)
printf('Possible short-circuit MVA at station 2 = %.2f MVA' ,SC_MVA2_poss)
```

Scilab code Exa 13.4 Example

```

// Variable Declaration
MVA_G1 = 20.0          //MVA rating of generator 1(MVA)
kv_G1 = 13.2           //Voltage rating of generator 1(kV)
x_G1 = 0.14            //Reactance of generator 1(p.u)
MVA_T1 = 20.0          //MVA rating of transformer 1(MVA)
kv_T1_lv = 13.2        //L.V voltage rating of transformer 1(kV)
kv_T1_hv = 132.0       //H.V voltage rating of transformer 1(kV)
x_T1 = 0.08            //Reactance of transformer 1(p.u)
MVA_G2 = 30.0          //MVA rating of generator 2(MVA)
kv_G2 = 13.2           //Voltage rating of generator 2(kV)
x_G2 = 0.16            //Reactance of generator 2(p.u)
MVA_T2 = 30.0          //MVA rating of transformer 2(MVA)
```

```
kv_T2_lv = 13.2      //L.V voltage rating of transformer 2(kV)
kv_T2_hv = 132.0     //H.V voltage rating of transformer 2(kV)
x_T2 = 0.12          //Reactance of transformer 2(p.u)
x_L = 75.0           //Line reactance(ohm)

// Calculation Section
MVA_base = 45.0                                     //Base MVA
kv_lv_base = 13.2                                   //L.T base voltage(kV)
kv_hv_base = 132.0                                  //H.T base voltage(kV)
I_lt_base = MVA_base*1000/(3**0.5*kv_lv_base)    //Base current on LT side(A)
x_G1_new = x_G1*(MVA_base/MVA_G1)                   //New reactance of generator 1(p.u)
x_G2_new = x_G2*(MVA_base/MVA_G2)                   //New reactance of generator 2(p.u)
x_T1_new = x_T1*(MVA_base/MVA_T1)                   //New reactance of transformer 1(p.u)
x_T2_new = x_T2*(MVA_base/MVA_T2)                   //New reactance of transformer 2(p.u)
x_L_new = x_L*(MVA_base/kv_hv_base**2)              //New line reactance(p.u)
V_f = 1.0                                           //Pre-fault voltage at fault point FF(p.u)
x_T = (x_L_new/2)+((x_G1_new+x_T1_new)*(x_G2_new+x_T2_new)/(x_G1_new+x_T1_new+x_G2_new+x_T2_new))  //Thevenin reactance(p.u)
I_f = V_f/complex(0,x_T)                                                        //Fault current(A)
I_G1 = I_f*(x_G2_new+x_T2_new)/(x_G1_new+x_T1_new+x_G2_new+x_T2_new)     //Fault current shared by generator 1(p.u)
```

```
I_f_G1 = I_G1*I_lt_base                  //Fault current shared by generator 1(A)
I_G2 = I_f*(x_G1_new+x_T1_new)/(x_G1_new+x_T1_new+x_G2_new+x_T2_new)     //Fault current shared by generator 2(p.u)
I_f_G2 = I_G2*I_lt_base                  //Fault current shared by generator 2(A)

// Result Section
printf('Fault current fed by generator 1 = %.1fj A',imag(I_f_G1))
printf('Fault current fed by generator 2 = %.1fj A',imag(I_f_G2))
printf('\nNOTE : ERROR : MVA ratings of G2 & T2 are 30 MVA , not 25 MVA as in textbook question')
```

Scilab code Exa 13.5 Example

```

// Variable Declaration
MVA_base = 20.0      //Base MVA

V_f = 1.0                              //Pre-fault voltage at bus 1(p.u).(Refer textbook diagram for marking.After circuit simplification)
x1 = 0.049                             //Reactance(p.u)
x2 = 0.064                             //Reactance(p.u)
x3 = 0.04                              //Reactance(p.u)

// Calculation Section
x_eq = (x1+x2)*x3/(x1+x2+x3)     //Equivalent reactance(p.u)
MVA_fault = V_f*MVA_base/x_eq    //Fault MVA
```

```


// Result Section
printf('SCC of bus 1 = %.f MVA' ,MVA_fault)
printf('\nNOTE : Changes in answer is due to more decimal places')
```

Scilab code Exa 13.6 Example

```

// Variable Declaration
x_G1 = 0.15                    //Sub-transient reactance of generator 1(p.u)
x_G2 = 0.15                    //Sub-transient reactance of generator 2(p.u)
x_T1 = 0.12                    //Leakage reactance of transformer 1(p.u)
x_T2 = 0.12                    //Leakage reactance of transformer 2(p.u)
x_s = 0.2                      //Reactance of tie line(p.u)
load = complex(1.5,0.5)        //Load(p.u)
S_12 = complex(0.75,0.25)      //Load at tie line(p.u)
V1 = 1.0                       //Pre-fault voltage at bus 1(p.u)

// Calculation Section
V_f = 1.0                                          //Voltage at FF(p.u)
Y_s = 1/complex(0,x_s)                             //Series admittance of line(p.u)
V2 = conj(1-(S_12/conj(Y_s)))      //Voltage at bus 2(p.u)
Z_L = conj(abs(V2)**2/load)                 //Load at bus 2(p.u)
```

```
I_s = (V1-V2)*Y_s                                    //Current through tie line(p.u)
I1 = I_s                                             //Current through G1 & T1(p.u)
I_L = V2/Z_L                                         //Load current(p.u)
I2 = I_L - I_s                                       //Pre-fault current from generator 2(p.u)

x_eq = (x_G1+x_T1)*(x_G2+x_T2+x_s)/(x_G1+x_T1+x_G2+x_T2+x_s)            //Equivalent reactance of n/w(p.u)
I_f = 1/complex(0,x_eq)                                                  //Fault current(p.u)
I_f1 = I_f*(x_G2+x_T2+x_s)/(x_G1+x_T1+x_G2+x_T2+x_s)                     //Fault current through G1,T1 towards F(p.u)
I_f2 = I_f*(x_G1+x_T1)/(x_G1+x_T1+x_G2+x_T2+x_s)                         //Fault current through G2,T2 & tie-line towards F(p.u)

V_1f = 0                                                                 //Post-fault voltage at bus 1(p.u)
V_2f = V_1f+(I_f2-I_s)*complex(0,x_s)                                    //Post-fault voltage at bus 2(p.u)

SCC = V_f/x_eq                                                           //Fault MVA or SCC

// Result Section
disp('Case(a) :')
printf('SCC of bus 1 = %.2f p.u',SCC)
disp('Case(b) :')
printf('Total post-fault ac current shared by
```

```
generator 1 , I_f1 = %.2fj p.u' ,imag(I_f1))
printf('Total post-fault ac current shared by generator 2 , I_f2 = %.2fj p.u' ,imag(I_f2))
disp('Case(c) :')
printf('Post-fault voltage of bus 2 , V_2f = %.3f  %.2f  p.u' ,abs(V_2f),phasemag(V_2f))
```

Scilab code Exa 13.7 Example

```

// Variable Declaration
I_a = 10.0*exp(%i*90*%pi/180)     //Line current(A)
I_b = 10.0*exp(%i*-90*%pi/180)    //Line current(A)
I_c = 10.0*exp(%i*0*%pi/180)      //Line current(A)

// Calculation Section
a = 1.0*exp(%i*120*%pi/180)       //Operator
I_a0 = 1.0/3*(I_a+I_b+I_c)                    //Zero-sequence component(A)
I_a1 = 1.0/3*(I_a+a*I_b+a**2*I_c)             //Positive-sequence component(A)
I_a2 = 1.0/3*(I_a+a**2*I_b+a*I_c)             //Negative-sequence component(A)

// Result Section
printf('Zero-sequence component , I_a0 = %.2f  %.f  A' ,abs(I_a0),phasemag(I_a0))
printf('Positive-sequence component , I_a1 = %.3f  %.f  A' ,abs(I_a1),phasemag(I_a1))
printf('Negative-sequence component , I_a2 = %.1f  %.f  A' ,abs(I_a2),phasemag(I_a2))
```

Scilab code Exa 13.8 Example

```
// Variable Declaration
kv = 13.2          //Voltage rating of generator(kV)
MVA = 25.0         //MVA rating of generator
MVA_sc = 170.0     //Short circuit MVA
x0 = 0.05          //Zero sequence reactance(p.u)
x2 = 0.13          //Negative sequence reactance(p.u)

MVA_base = 25.0                                   //Base MVA
kv_base = 13.2                                    //Line-to-line Base voltage(kV)
I_base = MVA_base*1000/(3**0.5*kv_base)           //Base current(A)
x1 = MVA_base/MVA_sc                              //Positive sequence reactance(p.u)
V_f = 1.0                                         //Pre-fault terminal voltage(p.u)
Z_f = 0                                           //Fault impedance
a = 1.0*exp(%i*120*%pi/180)             //Operator

// Calculation Section
I_a1 = V_f/complex(0,(x0+x1+x2))                  //Positive sequence current(p.u)
I_a2 = I_a1                                       //Negative sequence current(p.u)
I_a0 = I_a1                                       //Zero sequence current(p.u)
I_a = 3*I_a1*I_base                               //Fault current at phase a(A)
I_b = 0                                           //Fault current at phase b(A)
I_c = 0                                           //Fault current at phase c(A)
V_a1 = V_f - I_a1*complex(0,x1)                   //Terminal voltage(p.u)
V_a2 = -I_a2*complex(0,x2)                        //
```

```
    Terminal voltage(p.u)
V_a0 = -I_a0*complex(0,x0)                           //
    Terminal voltage(p.u)
V_a = (V_a0+V_a1+V_a2)*kv_base/3**0.5                //
    Line-to-neutral voltage at terminal(kV)
V_b = (V_a0+a**2*V_a1+a*V_a2)*kv_base/3**0.5         //
    Line-to-neutral voltage at terminal(kV)
V_c = (V_a0+a*V_a1+a**2*V_a2)*kv_base/3**0.5         //
    Line-to-neutral voltage at terminal(kV)
V_ab = (V_a-V_b)                                     //
    Line voltages at terminal(kV)
V_bc = (V_b-V_c)                                     //
    Line voltages at terminal(kV)
V_ca = (V_c-V_a)                                     //
    Line voltages at terminal(kV)

I_a12 = V_f/complex(0,(x1+x2))                         //
    Positive sequence current(p.u)
I_a22 = -I_a12                                         //
    Negative sequence current(p.u)
I_a02 = 0                                              //
    Zero sequence current(p.u)
I_a_2 = (I_a12+I_a22+I_a02)*I_base                     //
    Fault current at phase a(A)
I_b_2 = (a**2*I_a12+a*I_a22+I_a02)*I_base              //
    Fault current at phase b(A)
I_c_2 = -I_b_2                                         //
    Fault current at phase c(A)
V_a12 = V_f - I_a12*complex(0,x1)                      //
    Terminal voltage(p.u)
V_a22 = V_a12                                          //
    Terminal voltage(p.u)
V_a02 = 0                                              //
    Terminal voltage(p.u)
V_a_2 = (V_a02+V_a12+V_a22)*kv_base/3**0.5             //
    Line-to-neutral voltage at terminal(kV)
V_b_2 = (V_a02+a**2*V_a12+a*V_a22)*kv_base/3**0.5 //
    Line-to-neutral voltage at terminal(kV)
```

```
V_c_2 = (V_a02+a*V_a12+a**2*V_a22)*kv_base/3**0.5 // Line-to-neutral voltage at terminal(kV)
V_ab2 = (V_a_2-V_b_2) // Line voltages at terminal(kV)
V_bc2 = (V_b_2-V_c_2) // Line voltages at terminal(kV)
V_ca2 = (V_c_2-V_a_2) // Line voltages at terminal(kV)

I_a13 = V_f/complex(0,(x1+(x0*x2/(x0+x2)))) // Positive sequence current(p.u)
I_a23 = -I_a13*x0/(x0+x2) // Negative sequence current(p.u)
I_a03 = -I_a13*x2/(x0+x2) // Zero sequence current(p.u)
I_a_3 = (I_a13+I_a23+I_a03)*I_base // Fault current at phase a(A)
I_b_3 = (I_a03+a**2*I_a13+a*I_a23)*I_base // Fault current at phase b(A)
I_c_3 = (I_a03+a*I_a13+a**2*I_a23)*I_base // Fault current at phase c(A)
V_a13 = V_f-I_a13*complex(0,x1) // Terminal voltage(p.u)
V_a23 = V_a13 // Terminal voltage(p.u)
V_a03 = V_a13 // Terminal voltage(p.u)
V_a3 = (V_a03+V_a13+V_a23)*kv_base/3**0.5 // Line-to-neutral voltage at terminal(kV)
V_b3 = (V_a03+a**2*V_a13+a*V_a23)*kv_base/3**0.5 // Line-to-neutral voltage at terminal(kV)
V_c3 = (V_a03+a*V_a13+a**2*V_a23)*kv_base/3**0.5 // Line-to-neutral voltage at terminal(kV)
V_ab3 = (V_a3-V_b3) // Line voltages at terminal(kV)
V_bc3 = (V_b3-V_c3) // Line voltages at terminal(kV)
V_ca3 = (V_c3-V_a3) //
```

```
Line voltages at terminal(kV)


// Result Section
printf('Case(i) : L-G fault :')
printf('Short circuit current , I_a = %.1fj A = %.1f %.f A',imag(I_a),abs(I_a),phasemag(I_a))
printf('Short circuit current , I_b = %.f %.f A',abs(I_b),phasemag(I_b))
printf('Short circuit current , I_c = %.f %.f A',abs(I_c),phasemag(I_c))
printf('Terminal line voltage , V_ab = %.2f %.2f kV',abs(V_ab),phasemag(V_ab))
printf('Terminal line voltage , V_bc = %.2f %.2f kV',abs(V_bc),phasemag(V_bc))
printf('Terminal line voltage , V_ca = %.2f %.2f kV',abs(V_ca),phasemag(V_ca))
printf('\nCase(ii) : L-L fault :')
printf('Short circuit current , I_a = %.f %.f A',abs(I_a_2),phasemag(I_a_2))
printf('Short circuit current , I_b = %.2f %.1f A',abs(I_b_2),phasemag(I_b_2))
printf('Short circuit current , I_c = %.2f %.1f A',abs(I_c_2),phasemag(I_c_2))
printf('Terminal line voltage , V_ab = %.3f %.1f kV',abs(V_ab2),phasemag(V_ab2))
printf('Terminal line voltage , V_bc = %.f %.1f kV',abs(V_bc2),phasemag(V_bc2))
printf('Terminal line voltage , V_ca = %.3f %.1f kV',abs(V_ca2),phasemag(V_ca2))
printf('\nCase(iii) : L-L-G fault :')
printf('Short circuit current , I_a = %.f %.f A',abs(I_a_3),phasemag(I_a_3))
printf('Short circuit current , I_b = %.2f %.1f A',abs(I_b_3),phasemag(I_b_3))
printf('Short circuit current , I_c = %.2f %.1f A',abs(I_c_3),phasemag(I_c_3))
printf('Terminal line voltage , V_ab = %.3f %.f
```

```
kV',abs(V_ab3),phasemag(V_ab3))
printf('Terminal line voltage , V_bc = %.f  %.f kV',abs(V_bc3),phasemag(V_bc3))
printf('Terminal line voltage , V_ca = %.3f  %.f kV',abs(V_ca3),phasemag(V_ca3))
printf('\nNOTE : Changes in answer is due to more decimal places')
```

Scilab code Exa 13.9 Example

```

// Variable Declaration
x0 = 0.05        //Zero sequence reactance(p.u)
x2 = 0.13        //Negative sequence reactance(p.u)
r = 1.0          //Resistance through which generator neutral is earthed(ohm)
MVA_sc = 170.0   //Short circuit MVA

// Calculation Section
MVA_base = 25.0                          //Base MVA
kv_base = 13.2                           //Line-to-line Base voltage(kV)
I_base = MVA_base*1000/(3**0.5*kv_base)  //Base current(A)
kv_base1 = 11.0                          //Base kV
Z_n = r*MVA_base/kv_base1**2             //Neutral impedance(p.u)
V_f = 1.0                                //Pre-fault terminal voltage(p.u)
x1 = MVA_base/MVA_sc                     //Positive sequence reactance(p.u)
I_a1 = V_f/complex(3*Z_n,(x1+x2+x0))     //Positive sequence current(p.u)
I_a0 = I_a1                              //Zero sequence current(p.u)
```

```
I_a2 = I_a1                                        //Negative sequence current(p.u)
I_a = 3*I_a1*I_base                                //Fault current(A)
V_n = 3*I_a0*Z_n*I_base                            //Potential of neutral(V)

// Result Section
printf('Fault current for a L-G short-circuit at its terminals , I_a = %.2f∠%.2f A',abs(I_a),phasemag(I_a))
printf('Neutral potential = %.3f∠%.2f V',abs(V_n),phasemag(V_n))
printf('\nNOTE : ERROR : For calculating neutral potential in textbook Z_n = 1 is taken instead of Z_n = 0.206611570248')
```

Scilab code Exa 13.10 Example

```

// Variable Declaration
x1_G1 = complex(0,0.17)     //Positive sequence reactance of G1(p.u)
x2_G1 = complex(0,0.14)     //Negative sequence reactance of G1(p.u)
x0_G1 = complex(0,0.05)     //Zero sequence reactance of G1(p.u)
x1_G2 = complex(0,0.17)     //Positive sequence reactance of G2(p.u)
x2_G2 = complex(0,0.14)     //Negative sequence reactance of G2(p.u)
x0_G2 = complex(0,0.05)     //Zero sequence reactance of G2(p.u)
x1_T1 = complex(0,0.11)     //Positive sequence reactance of T1(p.u)
```

```
x2_T1 = complex(0,0.11)       //Negative sequence reactance of T1(p.u)
x0_T1 = complex(0,0.11)       //Zero sequence reactance of T1(p.u)
x1_T2 = complex(0,0.11)       //Positive sequence reactance of T2(p.u)
x2_T2 = complex(0,0.11)       //Negative sequence reactance of T2(p.u)
x0_T2 = complex(0,0.11)       //Zero sequence reactance of T2(p.u)
x1_L = complex(0,0.22)        //Positive sequence reactance of line(p.u)
x2_L = complex(0,0.22)        //Negative sequence reactance of line(p.u)
x0_L = complex(0,0.60)        //Zero sequence reactance of line(p.u)


// Calculation Section
a = 1.0*exp(%i*120*%pi/180)                                //Operator
Z_1T = (x1_G1+x1_T1)*(x1_G2+x1_T2+x1_L)/(x1_G1+x1_T1+x1_G2+x1_T2+x1_L)  //Thevenin reactance of positive sequence(p.u)
Z_2T = (x2_G1+x2_T1)*(x2_G2+x2_T2+x2_L)/(x2_G1+x2_T1+x2_G2+x2_T2+x2_L)  //Thevenin reactance of negative sequence(p.u)
Z_0T = (x0_G1+x0_T1)*(x0_T2+x0_L)/(x0_G1+x0_T1+x0_T2+x0_L)              //Thevenin reactance of zero sequence(p.u)
V_f = 1.0                                                   //Pre-fault terminal voltage(p.u)
I_a1 = V_f/(Z_1T+Z_2T+Z_0T)                                 //Positive sequence current(p.u)
I_a2 = I_a1
```

```
//Negative sequence current(p.u)
I_a0 = I_a1
//Zero sequence current(p.u)
I_a = 3*I_a1
//Fault current(p.u)

I_a1_G1 = I_a1*(x1_L+x1_T2+x1_G2)/(x1_L+x1_T1+x1_G1+x1_T2+x1_G2)        //Positive sequence current shared by G1(p.u)
I_a2_G1 = I_a2*(x2_L+x2_T2+x2_G2)/(x2_L+x2_T1+x2_G1+x2_T2+x2_G2)        //Negative sequence current shared by G1(p.u)
I_a0_G1 = I_a0*(x0_L+x0_T2)/(x0_L+x0_T1+x0_G1+x0_T2)                    //Zero sequence current shared by G1(p.u)
I_a_G1 = I_a0_G1+I_a1_G1+I_a2_G1                                        //Phase current through G1(p.u)
I_b_G1 = I_a0_G1+a**2*I_a1_G1+a*I_a2_G1                                 //Phase current through G1(p.u)
I_c_G1 = I_a0_G1+a*I_a1_G1+a**2*I_a2_G1                                 //Phase current through G1(p.u)

I_a1_G2 = I_a1*(x1_T1+x1_G1)/(x1_L+x1_T1+x1_G1+x1_T2+x1_G2)*exp(%i*30*%pi/180)    //Positive sequence current shared by G1(p.u)
I_a2_G2 = I_a2*(x2_T1+x2_G1)/(x2_L+x2_T1+x2_G1+x2_T2+x2_G2)*exp(%i*-30*%pi/180)   //Negative sequence current shared by G1(p.u)
I_a0_G2 = 0
//Zero sequence current shared by G1(p.u)
I_a_G2 = I_a0_G2+I_a1_G2+I_a2_G2
```

```
    //Phase current through G2(p.u)
I_b_G2 = I_a0_G2+a**2*I_a1_G2+a*I_a2_G2
    //Phase current through G2(p.u)
I_c_G2 = I_a0_G2+a*I_a1_G2+a**2*I_a2_G2
    //Phase current through G2(p.u)


// Result Section
printf('Fault current for a L-G fault at bus 1 , I_a = %.3fj p.u',imag(I_a))
printf('\nPhase currents contributed by G1 :')
printf('I_a = %.3f  %.1f   p.u',abs(I_a_G1),phasemag(I_a_G1))
printf('I_b = %.3f  %.1f   p.u',abs(I_b_G1),phasemag(I_b_G1))
printf('I_c = %.3f  %.1f   p.u',abs(I_c_G1),phasemag(I_c_G1))
printf('\nPhase currents contributed by G2 :')
printf('I_a = %.3f  %.1f   p.u',abs(I_a_G2),phasemag(I_a_G2))
printf('I_b = %.3f  %.1f   p.u',abs(I_b_G2),phasemag(I_b_G2))
printf('I_c = %.3f  %.1f   p.u',abs(I_c_G2),phasemag(I_c_G2))
printf('\nNOTE : ERROR : Calculation mistakes in Generator G2 part')
```

Scilab code Exa 13.11 Example

```


// Variable Declaration
kv_G1 = 13.2        //Voltage rating of G1(kV)
```

```
MVA_G1 = 40.0          //MVA rating of G1
x1_G1 = 0.2            //Positive sequence reactance of G1(p.u)
x2_G1 = 0.2            //Negative sequence reactance of G1(p.u)
x0_G1 = 0.08           //Zero sequence reactance of G1(p.u)
MVA_T1 = 40.0          //MVA rating of T1
x_T1 = 0.05            //Reactance(p.u)
kv_lv_T1 = 13.2        //L.V side rating of T1(kV)
kv_hv_T1 = 132.0       //H.V side rating of T1(kV)
kv_L = 132.0           //Voltage rating of line(kV)
x1_L = 40.0            //Positive sequence resistance of line(ohm)
x2_L = 40.0            //Negative sequence resistance of line(ohm)
x0_L = 100.0           //Zero sequence resistance of line(ohm)
MVA_T2 = 40.0          //MVA rating of T1
x_T2 = 1.0             //Resistance through which neutral is earthed(ohm)
xp_T2 = 0.05           //Primary reactance of T2(p.u)
xs_T2 = 0.045          //Secondary reactance of T2(p.u)
xt_T2 = 0.06           //Tertiary reactance of T2(p.u)

// Calculation Section
MVA_base = 40.0                                      //Base MVA
kv_base_G1 = 13.2                                    //Voltage base on generator side(kV)
kv_base_L = 132.0                                    //Voltage base on Line side(kV)
kv_base_T2t = 3.3                                    //Voltage base on tertiary side of T2(kV)
```

```
kv_base_T2s = 66                                         //Voltage base on secondary side of T2(kV)
R_ng = 2*MVA_base/kv_base_G1**2                          //Neutral resistance of generator(p.u)
x1_L_new = x1_L*MVA_base/kv_base_L**2                    //New Line reactance(p.u)
x2_L_new = x2_L*MVA_base/kv_base_L**2                    //New Line reactance(p.u)
x0_L_new = x0_L*MVA_base/kv_base_L**2                    //New Line reactance(p.u)
R_nT = x_T2*MVA_base/kv_base_T2s**2                      //Neutral resistance of T2(p.u)
V_f = 1.0                                                //Pre-fault voltage at fault point(p.u)
Z1 = complex(0,x1_G1+x_T1+(x1_L_new/2)+xp_T2+xs_T2)      //Thevenin impedance of positive sequence(p.u)
Z2 = complex(0,x2_G1+x_T1+(x2_L_new/2)+xp_T2+xs_T2)      //Thevenin impedance of negative sequence(p.u)
Z0 = complex(0.0024,0.0593)                              //Thevenin impedance of zero sequence(p.u).Refer diagram
I_f = 3*V_f/(Z1+Z2+Z0)                                   //Fault current(p.u)
I_f1 = abs(I_f)*MVA_base*1000/(3**0.5*kv_base_T2s)       //Fault current(A)
MVA_fault = abs(I_f)*MVA_base
```

```
    //Fault MVA

// Result Section
printf('Fault current , I_f = %.2f A' ,I_f1)
printf('Fault MVA for L-G fault = %.2f MVA' ,
    MVA_fault)
```

Chapter 14

ELEMENTS OF CIRCUIT BREAKERS AND RELAYS

Scilab code Exa 14.1 Example

```

// Variable Declaration
TMS = 0.5        //Time multiplier setting
I_f = 5000.0     //Fault current(A)
CT = 500.0/5     //CT ratio
set_plug = 1.0   //Relay plug set
I_relay = 5.0    //Rated relay current(A)

// Calculation Section
PSM = I_f/(CT*set_plug*I_relay)      //Plug setting multiplier
T1 = 1.0                             //Time of operation for obtained PSM & TMS of 1 from graph. Refer Fig 14.22
T2 = TMS*3/T1                        //Time of operation(sec)


// Result Section
```

```
printf('Operating time of the relay = %.1f sec',T2)
```

Scilab code Exa 14.2 Example

```

// Variable Declaration
I_f_A = 6000.0        //3-phase fault current of substation A(A)
I_f_B = 5000.0        //3-phase fault current of substation B(A)
I_f_C = 3000.0        //3-phase fault current of substation C(A)
I_f_D = 2000.0        //3-phase fault current of substation D(A)
I_L_max = 100.0       //Maximum load cuurent(A)
T = 0.5               //Operating time of breakers(sec)


I_set = 1.0                                //Setting current(A)

// Calculation Section
I_L_maxD = I_L_max                         //Maximum load current at D(A)
CT_D = I_L_max/1                           //CT ratio
PSM_D = I_f_D/(CT_D*I_set)                 //Plug setting multiplier
TMS_D = 0.1                                //Time multiplier setting
T_D = 0.14*TMS_D/(PSM_D**0.02-1)           //Time of operation(sec)

I_L_maxC = I_L_max+I_L_maxD                //Maximum load current at C(A)
```

```
CT_C = I_L_maxC/1                        //CT ratio
PSM_C = I_f_C/(CT_C*I_set)               //Plug setting multiplier
T_C = T_D+T                              //Minimum time of operation(sec)
TMS_C = T_C*(PSM_C**0.02-1)/0.14         //Time multiplier setting

I_L_maxB = I_L_max+I_L_maxC              //Maximum load current at B(A)
CT_B = I_L_maxB/1                        //CT ratio
PSM_B = I_f_B/(CT_B*I_set)               //Plug setting multiplier
T_B = T_C+T                              //Minimum time of operation(sec)
TMS_B = T_B*(PSM_B**0.02-1)/0.14         //Time multiplier setting

I_L_maxA = I_L_max+I_L_maxB              //Maximum load current at A(A)
CT_A = I_L_maxA/1                        //CT ratio
PSM_A = I_f_A/(CT_A*I_set)               //Plug setting multiplier
T_A = T_B+T                              //Minimum time of operation(sec)
TMS_A = T_A*(PSM_A**0.02-1)/0.14         //Time multiplier setting

// Result Section
printf('Relay A :')
printf('CT ratio = %.f/1',CT_A)
printf('PSM of R_A = %.1f',PSM_A)
printf('TMS of R_A = %.1f sec',TMS_A)
printf('\nRelay B :')
printf('CT ratio = %.f/1',CT_B)
printf('PSM of R_B = %.2f',PSM_B)
printf('TMS of R_B = %.1f sec',TMS_B)
printf('\nRelay C :')
```

```
printf('CT ratio = %.f/1',CT_C)
printf('PSM of R_C = %.1f',PSM_C)
printf('TMS of R_C = %.1f sec',TMS_C)
printf('\nRelay D :')
printf('CT ratio = %.f/1',CT_D)
printf('PSM of R_D = %.1f',PSM_D)
printf('TMS of R_D = %.2f sec',TMS_D)
```

Scilab code Exa 14.3 Example

```

// Variable Declaration
kv_hv = 66.0                              //Voltage rating of HV side of transformer(kV)
kv_lv = 11.0                              //Voltage rating of LV side of transformer(kV)
CT = 300.0/5                              //CT ratio on low tension side

// Calculation Section
I = 300.0                                 //Assumed current flowing at low tension side(A)
I_HT = kv_lv/kv_hv*I                      //Line current on HT side(A)
I_LT_CT = I/CT                            //Pilot wire current from LT side(A)
CT_ratio_HT = I_HT*3**0.5/I_LT_CT         //Ratio of CT on HT side


// Result Section
printf('Ratio of CT on high tension side = %.f  3 /%.f',I_HT,I_LT_CT)
```

Scilab code Exa 14.4 Example

```

// Variable Declaration
kv = 11.0         //Voltage rating(kV)
MVA = 5.0         //MVA rating
R = 10.0          //Resistance(ohm)
per_a = 0.15      //Armature winding reactance
per_trip = 0.3    //Relay trip for out-of-balance

// Calculation Section
x_p = per_a*kv**2/MVA                              //Winding Reactance(ohm)
V = kv/3**0.5*1000                                 //Phase voltage(V)
I = per_trip*MVA*1000/(3**0.5*kv)                  //Out of balance current(A)
p = (((R*I)**2/(V**2-(x_p*I)**2))**0.5)*100        //Percentage of winding remains unsupported

// Result Section
printf('Percentage of winding that remains unprotected , p = %.1f percentage' ,p)
```

Chapter 15

POWER SYSTEM STABILITY

Scilab code Exa 15.1 Example

```

// Variable Declaration
G = 50.0                  //Rating of machine(MVA)
f = 50.0                  //Frequency of turbo generator(
    Hz)
V = 11.0                  //Voltage rating of machine(kV)
H = 9.0                   //Cycle corresponding to 180 ms
P_0 = 40.0                //Pre-fault output power(MW)
delta_0 = 20.0            //Rotor angle at instant of
    fault(degree)

funcprot(0)
// Calculation Section
P_0_close = 0                                   //Output
    power at instant of reclosing(MW)
P_a = P_0 - P_0_close                           //Net
    accelerating power(MW)
delta_sqr = P_a*180*f/(G*H)                     //double
    derivative(elect.degrees/sec^2)
```

```
function ans =  integrand1(t)                            //intgs the double derivative to 800*t
    ans = delta_sqr
endfunction
a = intg(0, 180*10**-3,integrand1)    //Rotor velocity(electrical degrees/sec)

function ans = integrand2(t)                            //intgs the double derivative to 400*t^2
    ans = delta_sqr*t
endfunction
b = intg(0, 180*10**-3,integrand2)
delta = delta_0 + b                                     //Rotor angle(electrical degrees)

// Result Section
printf('Rotor angle at the instant of reclosure = %.2f electrical degrees',delta)
printf('Rotor velocity at the instant of reclosure = %.1f electrical degrees/sec',a)
```

Scilab code Exa 15.2 Example

```

// Variable Declaration
V = 1.0         //Infinite bus voltage(p.u)
E = 1.0         //e.m.f of finite generator behind transient reactance(p.u)
X_T = 0.8       //Transfer reactance(p.u)
P_i = 0.5       //Input power(p.u)
P_i_d = 0.8     //p.u
```

```
P_0 = 0.5          //Output power(p.u)
P = 0.5            //Power(p.u)

// Calculation Section
P_m = E*V/X_T                         //Amplitude of power angle curve(p.u)
delta_0 = asin(P_i/P_m)         //Radians
delta = asin(P_i_d/P_m)         //Radians
delta_m = %pi-delta              //Radians
A_acc = P_i_d*(delta-delta_0)-P_m*(cos(delta_0)-cos(delta))    //Possible area of a// Result Sectioneleration
A_dec = P_m*(cos(delta)-cos(delta_m))-P_i_d*(delta_m-delta)    //Possible area of deceleration

// Result Section
if (A_acc < A_dec) then
    printf('System is stable')
    stability = A_dec/A_acc
    printf('Margin of stability = %.2f' ,stability)
else
    printf('System is not stable')
end
```

Scilab code Exa 15.3 Example

```

// Variable Declaration
x = 0.25           //Transient reactance(p.u)
E = 1.0            //e.m.f of finite generator behind transient reactance(p.u)
x_T = 0.1          //Reactance of transformer(p.u)
x_L = 0.4          //Reactance of one line(p.u)
P_i = 0.25         //Pre-fault power(p.u)

```

```
// Calculation Section
X_T = x+x_T+(x_L/2)                    //Transfer reactance at pre-fault state(p.u)
P_m = E**2/X_T                         //Amplitude of power angle curve at pre-fault state(p.u)
X_T1 = 1.45                            //Transfer reactance b/w finite generator & infinite bus at faulty state(p.u).Refer texbook problem for figure
P_m1 = E**2/X_T1                       //Amplitude of power angle curve at faulty state(p.u)
r1 = X_T/X_T1
delta_0 = asin(P_i/P_m)         //Radians
delta_1 = asin(P_i/(r1*P_m))    //Radians
delta_m = %pi - delta_1          //Radians

function ans = integrand1(delta)
    ans = r1*P_m*sin(delta)
endfunction
a = intg(delta_0, delta_1,integrand1)

A_acc = P_i*(delta_1-delta_0) - a

function ans = integrand2(delta)
    ans = r1*P_m*sin(delta)
endfunction

b = intg( delta_1, delta_m,integrand2)
A_dec = b - P_i*(delta_m-delta_1)
limit = 0.5648                         //Obtained by iterations.Refer textbook.Here assigned directly.


// Result Section
if(A_acc < A_dec) then
    printf('System is Stable')
    stability = A_dec/A_acc
    printf('Margin of stability = %.2f' ,stability)
```

```
else
    printf('System is not stable')
end
printf('Transient stability limit = %.4f p.u' ,limit
    )
printf('\nNOTE : ERROR : angle delta_0 = 7.9   =
    0.13788 radian not 0.014 radian as in textbook')
```

Scilab code Exa 15.4 Example

```


// Variable Declaration
x = 0.25          //Transient reactance(p.u)
E = 1.0           //e.m.f of finite generator behind
    transient reactance(p.u)
x_T = 0.1         //Reactance of transformer(p.u)
x_L = 0.4         //Reactance of one line(p.u)
P_i = 0.7         //Pre-fault power(p.u)

// Calculation Section
X_T = x+x_T+(x_L/2)                    //Transfer
    reactance at pre-fault state(p.u)
P_m = E**2/X_T                         //Amplitude of
    power angle curve at pre-fault state(p.u)
X_T1 = 1.45                            //Transfer
    reactance b/w finite generator & infinite bus at
    faulty state(p.u).Refer texbook problem for
    figure
P_m1 = E**2/X_T1                       //Amplitude of
    power angle curve at faulty state(p.u)
r1 = X_T/X_T1
X_T2 = x+x_T+x_L                       //Transfer
    reactance for post fault state(p.u)
r2 = X_T/X_T2
```

```
P_m2 = r2*P_m
delta_0 = asin(P_i/P_m)         //Radians
delta_1 = asin(P_i/(r2*P_m))    //Radians
delta_m = %pi - delta_1          //Radians
delta_c = 0.7                        //Specified value(radians)

function ans = integrand1(delta)
    ans = r1*P_m*sin(delta)
endfunction
a = intg(delta_0, delta_c,integrand1)

A_acc = P_i*(delta_c-delta_0) - a

function ans = integrand2(delta)
    ans = r2*P_m*sin(delta)
endfunction

b = intg(delta_c, delta_m,integrand2)
A_dec = b - P_i*(delta_m-delta_c)
cos_delta_cr = ((delta_m-delta_0)*sin(delta_0)-r1*cos(delta_0)+r2*cos(delta_m))/(r2-r1)
delta_cr = acos(cos_delta_cr)*180/%pi

// Result Section
if(A_acc < A_dec) then
    printf('System is Stable')
    stability = A_dec/A_acc
    printf('Margin of stability , K = %.2f' , stability)
else
    printf('System is not stable')
end
printf('Critical clearing angle for a certain pre-fault power = %.2f  ' ,delta_cr)
printf('Critical clearing time will be known from circuit-breaker specifications')
```

Scilab code Exa 15.5 Example

```

// Variable Declaration
P_i = 0.75        //Pre-fault power(p.u)
f = 50.0          //Frequency(Hz)
H = 6.0           //Value of H for finite machine(sec)
x_G = 0.2         //Reactance of machine(p.u)
x_T = 0.1         //Reactance of transformer(p.u)
x_L = 0.4         //Reactance of line(p.u)
V = 1.0           //Voltage of infinite bus(p.u)
E = 1.0           //e.m.f of finite generator behind transient reactance(p.u)

// Calculation Section
X_T = x_G+x_T+(x_L)                                   //Transfer reactance at pre-fault state(p.u)
P_m = E**2/X_T                                        //Amplitude of power angle curve at pre-fault state(p.u)
delta_0 = asin(P_i/P_m)                          //Radians
delta_0a = delta_0*180/%pi
delta_cr = acos((%pi-2*delta_0)*sin(delta_0)-cos(delta_0))
delta_cra = delta_cr*180/%pi
t_cr = ((delta_cra-delta_0a)*2*H/(180*f*P_i))**0.5

// Result Section
printf('Critical clearing angle for circuit breaker at bus 1 = %.2f ',delta_cra)
printf('Time for circuit breaker at bus 1 ,t_cr = %.3f sec',t_cr)
```